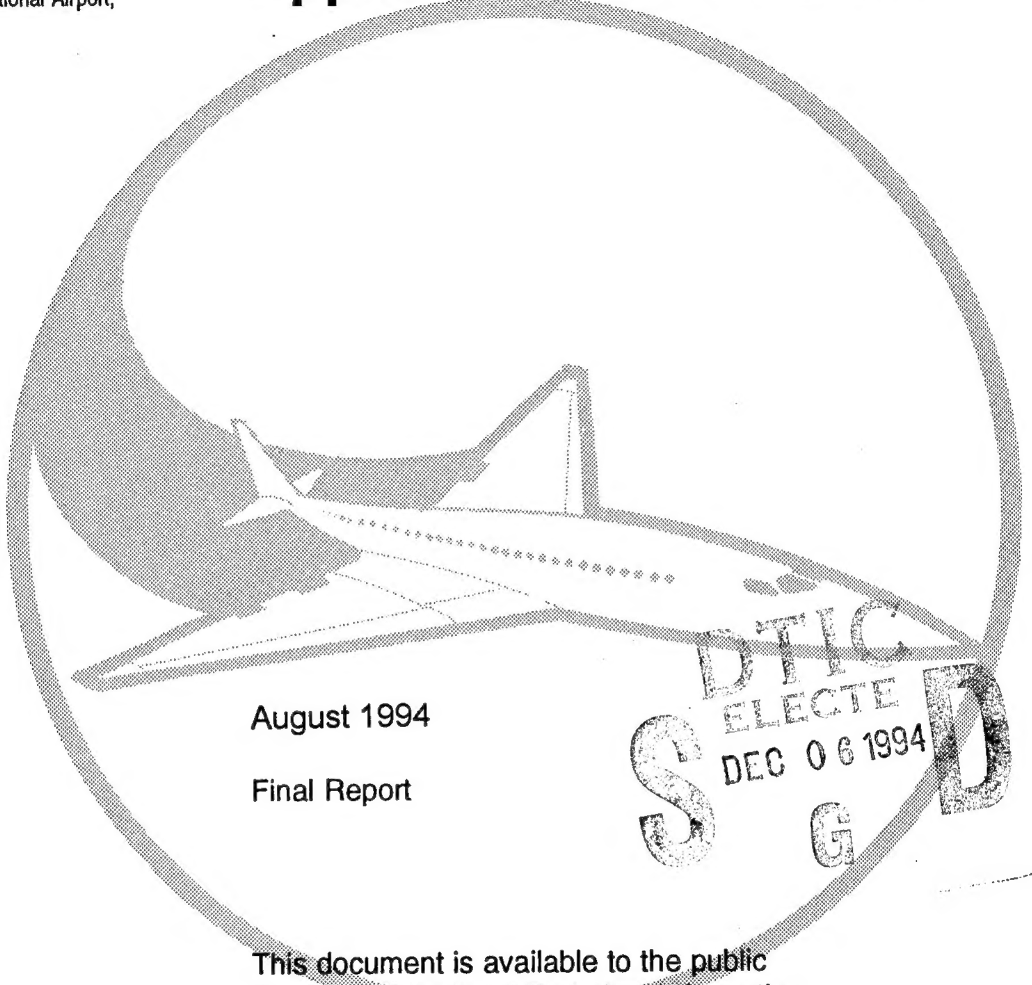
DOT/FAA/CT-94/41

FAA Technical Center
Atlantic City International Airport,
N.J. 08405

Artificial Intelligence With Applications for Aircraft



August 1994

Final Report

This document is available to the public through the National Technical Information Service, Springfield, Virginia 22161.

U.S. Department of Transportation
Federal Aviation Administration

Technical Report Documentation Page

1. Report No. DOT/FAA/CT-94/41	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle ARTIFICIAL INTELLIGENCE WITH APPLICATIONS FOR AIRCRAFT		5. Report Date August 1994 6. Performing Organization Code
7. Author(s) L. Harrison, P. Saunders, J. Janowitz		8. Performing Organization Report No.
9. Performing Organization Name and Address Galaxy Scientific Corporation 2500 English Creek Avenue Building 11 Pleasantville, NJ 08232		10. Work Unit No. (TRAIS) 11. Contract or Grant No. DTFAO3-89-C-00043
12. Sponsoring Agency Name and Address U.S. Department of Transportation Federal Aviation Administration Technical Center Atlantic City International Airport, NJ 08405		13. Type of Report and Period Covered Final Report 14. Sponsoring Agency Code ACD-230

15. Supplementary Notes

Pete Saraceni, FAA Technical Center, Program Manager, (609) 485-5577

16. Abstract

This report provides an overview of Artificial Intelligence (AI) technology, one of the more complex applications of digital systems. This report examines AI-based technology, focusing on three fields: Neural Networks, fuzzy logic, and Expert Systems. This report provides the reader with the background and a basic understanding of the fundamentals of these fields. Another section examines aspects of the AI development environment, including languages, tools, and AI-based hardware components.

Some of the proposed aviation-related applications for both civil and military aircraft, including pilot assistants and diagnostic aids, are surveyed. Additionally, certification issues, including regulations, guidelines, and verification and validation techniques are examined. Human factors issues relating to the use of this technology are identified and reviewed. In addition, the report identifies safety issues and concerns over the use of this technology in airborne systems.

17. Key Words Artificial Intelligence, avionics, blackboard system, certification, chaining, database, development tool, Expert System, fuzzy logic, guidelines, human factors, inference engine, knowledge base, learning, Neural Network, rule base, search technique, shell, validation, verification.	18. Distribution Statement This document is available to the public through the National Technical Information Service (NTIS), Springfield, Virginia 22161

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 179	22. Price

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

ACRONYMS

μC	micro Controller
μm	micrometer
AC	Advisory Circular
ACO	Aircraft Certification Office
AF	Activation Framework
AFO	Activation Framework Object
AGARD	Advisory Group for Aerospace Research and Development
AI	Artificial Intelligence
AIAA	American Institute of Aeronautics and Astronautics
AIM	Airmen's Information Manual
ANSI	American National Standards Institute
ARINC	Aeronautical Radio, Incorporated
ARP	Aerospace Recommended Practice
ARPA	Advanced Research Projects Agency (formerly DARPA)
ASIC	Application Specific Integrated Circuit
ATC	Air Traffic Control
ATE	Automatic Test Equipment
BABBAGE	Boeing Advanced BlackBoard Ada Generation Environment
BIT	Built-In Test
CASEy	Connector Assembly Specifications Expert
CASSY	Cockpit Assistant System
CDU	Control Display Unit
CE	Certification Engineer
CF	Certainty Factor
CM	Configuration Management
CMOS	Complimentary Metal-Oxide Semiconductor
CNI	Communication, Navigation, and Identification
CPU	Central Processing Unit
CRTIE	Common Run-Time Inference Engine
CWT	Continuous Wavelet Transformation
DOD	Department of Defense
DOS	Disk Operating System
DOT	Department of Transportation
DSP	Digital Signal Processor
EEPROM	Electrically Erasable Programmable Read Only Memory
EPES	Emergency Procedures Expert System
ES	Expert System
ETANN	Electrically Trainable Analog Neural Network
F/FA	Fault and Failure Analysis
FAA	Federal Aviation Administration
FAR	Federal Aviation Regulation
FCA	Fuzzy Computational Accelerator
FFT	Fast Fourier Transform

IDE	Fuzzy Inference Development Environment
FIL	Fuzzy Inference Language
FINDER	Flight-Plan Interactive Negotiation and Decision-Aiding System for Enroute Rerouting
FLEX	FORTRAN Library for EXpert systems
FSSE	Fuzzy System Standard Environment
FTPP	Fault Tolerant Parallel Processor
GaAs	gallium arsenide
GUI	Graphical User Interface
HF	High Frequency
HOL	High-Order Language
IC	Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
IETM	Interactive Electronic Technical Manual
IJCNN	International Joint Conference on Neural Networks
I/O	Input/Output
IV&V	Independent Verification and Validation
k	thousand
KBAP	Knowledge-Based AutoPilot
LED	Light Emitting Diode
LISP	LISt Processor
LRU	Line Replaceable Unit
MFLOPS	Million Floating Point Operations per Second
MHz	MegaHertz
mm	millimeter
NASA	National Aeronautics and Space Administration
NN	Neural Network
ONR	Office of Naval Research
Op Amp	Operational Amplifier
PA	Pilot's Associate
PC	personal computer
PE	Processing Element
PGA	pin grid array
PGG	Plan-Goal Graph
PMA	Portable Maintenance Aid
PROLOG	Programming in Logic
PVI	Pilot Vehicle Interface
QA	Quality Assurance
RAE	Royal Aerospace Establishment
RISC	Reduced Instruction Set Computer
RPA	Rotorcraft Pilot's Associate
RTCA	Requirements and Technical Concepts for Aviation (formerly Radio Technical Commission for Aeronautics)
SAE	Engineering Society for Advancing Mobility Land Sea Air and Space (formerly Society of Automotive Engineers)
SC	Special Committee
S/W	Software

STC	Supplemental Type Certificate
TACAID	Tactical Aid
TC	Type Certificate
TCAS	Traffic alert and Collision Avoidance System
TEAMS	Technical Expert Aircraft Maintenance System
TSO	Technical Standing Order
TTL	Transistor-Transistor Logic
UHF	Ultra High Frequency
USAF	United States Air Force
UUT	Unit Under Test
VHF	Very High Frequency
V&V	Verification and Validation
VLSI	Very Large Scale Integration

EXECUTIVE SUMMARY

This technical report was initiated by the Federal Aviation Administration Technical Center's Directorate for Aircraft Safety, Flight Safety Research Branch. The purpose of the report is to provide an overview of Artificial Intelligence (AI) technology, highlighting selected fields. This report also forms the basis for chapter 20 of the Digital Systems Validation Handbook – Volume II. The Digital Systems Validation Handbook provides guidance in new technologies to certification specialists. The report presents information in a way that enables the certification specialist to understand the information presented in type certification and supplemental type certification and to discuss this information with the design engineer.

One of the more complex applications of digital systems is in the field of AI. This report examines AI-based technology, the development environment, and proposed aviation-related applications of this technology. In addition, the report identifies safety issues and concerns over the use of this technology in airborne systems. There are seven sections of the report:

- Section 1 – Introduction
- Section 2 – Artificial Intelligence Overview
- Section 3 – Artificial Intelligence and Expert System Development
- Section 4 – Artificial Intelligence and Expert System Applications
- Section 5 – Certification Considerations for Artificial Intelligence
- Section 6 – Human Factors and Artificial Intelligence
- Section 7 – Conclusion

While AI-based technology may never be able to replace genuine intelligence, it can provide users with many benefits. Developing human-centered automation and designing advanced technology that will capitalize on the relative strengths of humans and machines are key to the success and usefulness of AI. For example, Expert Systems (ESs) can consistently provide expert advice in a timely manner. ESs are not influenced by factors which impair human decision making, such as stress.

Neural Network systems are helpful for applications such as pattern recognition. Recognizing pre-failure signatures in airborne machinery is an excellent application for this particular field of AI. Fuzzy logic systems are useful for applications that normally require human intuition, are difficult to control with conventional techniques, or are difficult to model.

One of the primary activities that requires addressing by developers and certification specialists concerns Verification and Validation (V&V) methods for AI-based systems. Numerous V&V techniques that have been successfully applied to ESs are identified in this report. Identifying a set of tests that can be demonstrated to satisfy safety requirements will be a challenge.

AI technology is not efficient at solving all types of problems. For a number of applications, it can, however, assist in managing problem complexity. AI-based systems for commercial aircraft are being researched and tested. Ultimately, such systems can offer economic advantages and contribute to flight safety.

1. INTRODUCTION.

1.1 DEFINING ARTIFICIAL INTELLIGENCE.

Because of ongoing debate over what comprises Artificial Intelligence (AI) technology and what the research goals are, it is difficult to provide an unequivocal definition of AI. Should some fields of AI technology, such as Expert Systems (ESs) or fuzzy logic, be considered simply another software application? Should a system without learning capability be considered an AI-based system, as some believe? Others contend that due to fundamental differences between computers and people, the ultimate goal of the autonomous "thinking machine" will never be realized. Human qualities, such as intuition, can not be reproduced in the confines of silicon and software (Dreyfus and Dreyfus 1986).

Debate over these issues will continue. Numerous definitions of what comprises AI technology exist. However, for the purpose of this report, AI is defined as the subfield of computer science that attempts to use computers to emulate the way humans think and reason when solving problems. This agrees with Wickens (1992):

> The study of AI is devoted to developing computer programs that will mimic the product of intelligent human problem solving, perception, and thought.

Defined in this manner, AI includes systems that are capable of "learning," as well as those that have static rules and databases. The part of the definition that is stressed, and that most agree with, is the mimicking, or emulating, of human techniques.

In this report, particular emphasis is placed on three fields of AI technology: ESs, fuzzy logic, and Neural Networks (NNs). Aviation- and avionics-related applications, or proposed applications, have been numerous for these disciplines, but especially for the ES. Therefore, sections 4, on AI applications, and 5, on certification issues, focus on these fields of AI technology.

1.2 ARTIFICIAL INTELLIGENCE AND AVIATION APPLICATIONS.

AI technology has existed for a number of years. During the early years, the promises made concerning this technology did not match the products delivered. With this in mind, Bouchard (1991) said:

> The user community has been stung with high development costs and false expectations, although that is changing with respect to expert systems.

Some of the early applications of AI-based technology have included image and speech recognition systems, natural language systems, and handwriting analysis. While research in these fields continues, AI-based technology is being used in a number of other applications. AI-based systems are now being used for control and monitoring systems, financial analysis, medical prognosis, manufacturing, training, sorting through large amounts of data in databases, and scheduling. A great amount of research is being performed and sponsored by the National Aeronautics and Space Administration (NASA) for space applications, as well as for potential flightdeck applications for both the commercial and military communities. Among the Department of Defense (DOD) community, this technology is referred to as "Machine Intelligence."

AI-based systems are also being used by the Nuclear Power Industry (Naser 1991). Some of the benefits of such systems include consistency of reasoning in stressful situations, reduction in time to perform certain tasks, and prevention of equipment failures using predictive diagnostics. During the Gulf war, AI-based technology was used for the first time in a combat situation (Bouchard 1991).

As research in the field of AI progresses, AI technology continues to mature and emphasize realistic expectations. For instance, the Institute of Electrical and Electronics Engineers (IEEE) sponsors the Artificial Intelligence for Applications conference. This conference solicits papers describing how AI techniques have been used to solve significant problems. Papers are also invited that describe how AI techniques can be applied to problems of increasing complexity (IEEE Expert June 1992).

AI-based products that save the user time and money and outperform conventional software-based systems are currently being manufactured. According to Curran (1992), it is likely that AI and ES technology will be used in the near future in at least two avionic areas. For the aviation community, the possible benefits of using AI-based systems would include the following:

- Optimizing the use of airspace
- Reducing the cost of flying
- Meeting Air Traffic Control (ATC) requirements
- Aiding the decision making process of the flight crew
- Aiding maintenance activity
- Assisting data management

The amount of data that the flight crew must absorb has increased, due to the expanding complexity and number of avionic systems in the cockpit. Reducing pilot and information overload can be achieved using AI. As with any technology, the final system interface (i.e., man-machine interface) needs to be an integral part of the design and development cycle. The Federal Aviation Administration's (FAA) National Plan for Aviation Human Factors addresses this issue.

AI-based pilot aid systems are being designed by both military and commercial manufacturers. In Germany, the Cockpit Assistant System (CASSY), an AI-based pilot aid system, is undergoing flight testing, with the expectation that it will be operational as early as 1997 (Nordwall 1992). This system may use AI-based technologies, such as NNs, ESs, and fuzzy logic (Prevot, Onken, and Dudek 1991). Certification authorities in Europe and the United States should be cognizant and concerned over such activity. Above all, they should be prepared for the presentation of such an AI-based system for certification by an avionics or airframe manufacturer.

1.3 ARTIFICIAL INTELLIGENCE REPORT OVERVIEW.

This report consists of five main sections, as follows:

- Section 2 – Artificial Intelligence Overview
- Section 3 – Artificial Intelligence and Expert System Development
- Section 4 – Artificial Intelligence and Expert System Applications
- Section 5 – Certification and Artificial Intelligence
- Section 6 – Human Factors and Artificial Intelligence

Section 2 gives a general overview of AI technology and contains a brief history of AI. The focus of section 2 is placed upon ESs, fuzzy logic, and NNs.

Among topics discussed for ESs are basic theory, architecture, implementation, reasoning, blackboard systems, knowledge representation, and knowledge acquisition. The basic theory of fuzzy logic is examined, along with some design issues, advantages, and disadvantages. For NNs, the different types of NNs are examined, along with learning and transfer functions.

The advantages and disadvantages of these three fields of AI technology are also discussed. Additionally, integrated AI systems are examined. Among these systems are ESs integrated with NNs, ESs integrated with fuzzy logic, and fuzzy logic integrated with NNs.

Section 3 examines the software, hardware, and development environment related to the field of AI. Many issues should be examined when developers are considering a tool for building an AI-based system. Due to the availability of numerous tools, their options and suitability should be examined carefully.

Section 4 examines some of the AI-based applications proposed for the commercial and military cockpit, for fixed wing, as well as rotorcraft. These include flight management systems, navigation systems, decision support systems, and intelligent monitoring and diagnostic systems. Also, some of the design considerations for avionic applications are identified.

Section 5 examines various issues relating to the certification of AI-based systems. Relevant regulations and guidelines are identified. Conventional software and ES Verification and Validation (V&V) methods are examined. Other issues examined include design, certification, and supportability. Topics included in this section are of importance to certification specialists. V&V of AI-based systems is a topic identified for further research (DOT/FAA/CT-93/16).

Section 6 discusses some of the human factors concerns relating to the proposed use of AI.

AI is a broad discipline and this report is limited in scope and depth of coverage. For further information, the reader is urged to make use of the bibliography located after section 7. The bibliography includes magazines and organizations dedicated to AI theory and applications.

2. ARTIFICIAL INTELLIGENCE OVERVIEW.

As stated in the introduction, AI refers to the subfield of computer science that studies the process of programming machines to perform in a manner resembling human behavior. It is the computer-based solution to complex problems using the application of processes that are analogous to the human reasoning process (Rolston 1988).

Many disciplines contribute ideas to the field of AI. Basic ideas in fields such as mathematics, psychology, linguistics, philosophy, computer science, and engineering find frequent application in AI. Intelligence requires many strengths and many of the problems faced in established disciplines intersect the natural concerns of AI-related problems.

Two of the objectives of AI are to make computers more useful and to understand the principles that make intelligence possible in living beings. As the world grows more complex, AI-based computers can assist with many tasks. Resources such as energy, food, and brain power must be used as wisely as possible. High quality help from computers in ordinary computing, as well as computing that exhibits intelligence, can assist in this area.

The ultimate goal of AI research is to build a "person." This person will have various modules that will be integrated to produce the final system. These modules involve a number of different applications relevant to AI research. The input to the system includes vision and language interfaces with the external domain. The output of the system entails robotics, speech, and screen interaction with the external domain. The internal systems include deduction, search, planning, explanation, and learning (Charniak and McDermott 1985).

Intelligence requires more than the ability to reason; it also requires a great deal of knowledge about the world. The knowledge about the world needs to be encoded in a way that the computer system will be able to interpret. There are some properties inherent in knowledge that cause difficulty for AI-related purposes. Knowledge is difficult to characterize accurately; it is constantly changing; it is virtually unbounded; and it generally needs to be organized according to the way it will be used (Rich and Knight 1991).

Robotics is a significant subarea of AI. There are many possible uses for robots. Tasks such as welding, general house cleaning, construction, and making deliveries would be appropriate for a robot. Robotics is a complex area involving aspects other than AI, such as mechanical and electrical design. Successful implementation of a fully operational, independent robot has not been achieved. Tasks that are performed almost unconsciously by humans require many of the same abilities used in solving more intellectually demanding problems. Complex robot control problems require planning at high levels of abstraction, ignoring details, and then planning at lower and lower levels where details become important. Research into the area of robotics has helped to develop many other AI areas.

Interfacing with the user requires capabilities such as voice and visual interface. Further investigation into these areas is required for them to be useful to an AI system. An area of AI that requires more research and development is natural language comprehension. This is the ability of an AI system to interpret written or spoken words. Human communication employs complex and little understood processes almost effortlessly. The primary use of language is to transmit a piece of mental structure from one brain to another under circumstances in which each brain possesses large, highly similar surrounding mental structures that serve as a common context. A computer system capable of understanding a message in natural language would require both the contextual knowledge and the processes for making the inferences assumed by the message generator (Nilsson 1980).

Common sense reasoning involves reasoning about physical objects and their relationships to each other, as well as reasoning about actions and their consequences. For example, an object can only be in one place at a time, and if an object is dropped it will fall to the floor and may break. This reasoning is difficult to capture because it is taken for granted. People do not need to think about actions related to common sense reasoning; they are virtually instinctive. At this point in AI development, the domains that require only specialized expertise without the assistance of common sense knowledge are the AI systems that are flourishing as useful systems.

Another area of AI research is intelligent retrieval from databases. Database systems are computer systems that store a large body of facts about some subject in such a way that the data can be used to answer questions about that subject. Many techniques have been developed to enable the efficient representation, storage, and retrieval of large numbers of facts. These techniques become related to AI when the task requires retrieving answers that entail deductive reasoning from the facts in the database. The designer of an intelligent database retrieval system is faced with several problems, including understanding queries that are stated in natural language, deducing answers from stored facts, and acquiring and utilizing common knowledge.

Expert consulting systems are systems that provide users with expert conclusions about specialized subject areas. These systems include medical diagnostic systems, which, given the symptoms of the patient, may suggest tests to perform and, possibly, speculate what the disease may be (Charniak and McDermott 1985). The key problem in developing expert consulting systems is how to represent and use the knowledge that experts possess and use. This knowledge is often imprecise or uncertain.

The task of a computer generating a computer program is related to both theorem proving and robotics. The system takes in a description of what the program is supposed to do and then writes the code. Program verification and debugging are a few of the capabilities that such a system must achieve.

Specifying optimal schedules and combinations is another class of problems that can be solved using AI techniques. The effort is in making the time-versus-problem-size curve grow as slowly as possible. Several methods have been developed for delaying and moderating the inevitable combinatorial explosion. Knowledge about the problem domain is the key to more efficient solution methods.

Research is being done in the area of a computer's ability to perceive surroundings. It would be useful for computers to be capable of understanding their surroundings by hearing and seeing them. Understanding surroundings requires a large database of knowledge about the things being perceived. The ultimate goal is to represent the scene by a natural language description. The process of hypothesis formation requires a large amount of knowledge about the expected scenes. For example, when a robot enters a room, there will be some information in the AI system about what normally can be expected to be found in a room, such as windows, doors, and furniture.

A task that may be appropriate for an AI system is proving or disproving a theorem. This requires intuitive skills, such as guessing which theorems should be proved first to prove the main theorem and the ability to make deductions from hypotheses. There are many informal tasks that can be formalized as theorem proving problems. Therefore, this is an important area in the study of AI methods (Nilsson 1980).

There are AI programs that can perform a variety of tasks, such as solving mathematical problems, predicting the presence of mineral deposits, verifying the designs of electronic components, and using television cameras to see the world and identify what is there. AI programs are also capable of playing backgammon, controlling manufacturing processes, diagnosing computer faults, designing computers, underwriting insurance, playing chess, and much more (Charniak and McDermott 1985). More examples of the task domains of AI are shown in table 2.1-1.

TABLE 2.1-1. EXAMPLES OF TASK DOMAINS OF ARTIFICIAL INTELLIGENCE
(Rich and Knight 1991)

Mundane Tasks
• Perception - Vision, Speech
• Natural Language - Understanding, Generation, Translation
• Common Sense Reasoning
• Robot Control
Formal Tasks
• Games - Chess, Backgammon, Checkers, Go
• Mathematics - Geometry, Logic, Integral Calculus, Proving Properties of Programs
Expert Tasks
• Engineering - Design, Fault Finding, Manufacturing Planning
• Scientific Analysis
• Medical Diagnosis
• Financial Analysis

Three subfields of AI are examined in detail in the following sections. They are ESs, fuzzy logic, and NNs. ESs are programmed to use the knowledge of experts applied to specific task areas. ESs can achieve high levels of system performance in limited domains. Fuzzy logic is used to model systems where information may be imprecise or incomplete. Fuzzy logic is useful for systems that normally require human intuition, are difficult to control with conventional techniques, or are difficult to model. NNs are a simplified model of the brain's structure. They are used to solve problems that require input to output data mapping.

2.1 HISTORY.

Beginning in the 1960s, there was much interest in developing intelligent systems. Many promises were made regarding the future capabilities of computers. However, by the early 1970s, most of the promises that were made had not been met. AI research lost most of its funding and was regarded as an eccentric branch of computer science. People became skeptical of AI because of unrealistic claims and expectations. Interest and activity in the area of AI research grew again, beginning in the late 1970s and peaking in the mid-1980s. Fuzzy logic was developed in 1965 as a method of capturing human expertise and incorporating it into ESs. There is still a significant amount of research being done in the field today.

Initially, the aim of AI was to develop universal models for general problem solving. However, soon it became apparent that this would rule out too many specific types of problems (Adeli 1990). Compared to other branches of engineering and computer science, AI is a very young field. Scientists and engineers are still struggling to find its rightful place in the scientific world. Although AI may not live up to early expectations, it can still benefit many applications.

2.2 EXPERT SYSTEMS.

An ES is a computer-based system designed to emulate the problem solving behavior of a human who is an expert in a narrow domain. ESs are based on AI techniques that use knowledge about a particular domain and reasoning techniques to perform activities normally done by human experts. They are created by first capturing the domain expert's knowledge and then translating and storing this knowledge in a computer readable format, discussed in more detail below. ESs emulate the reasoning process of a human expert, sometimes reasoning from imprecise and uncertain information. They solve real-world, complex problems, using a computer model of expert human reasoning. These systems provide a way to make the knowledge of experts available to many.

Research in ESs began in the middle 1960s. There is an extensive body of knowledge covering a number of different approaches to ESs. Research and development for these systems is continuing at a rapid pace. There are many ESs, either in use or under development, for providing assistance in diverse applications such as:

- Performance monitoring and diagnosis of telecommunication systems
- Soil analysis
- Decision aiding for military maneuvers
- Data Quality Assurance (QA) in the Consumer Price Index
- Computer hardware configuration for large computer systems

A number of other applications related to avionics are examined in section 4.

Research in the area of languages to support symbolic reasoning has been done in conjunction with AI and ES research. LISt Processor (LISP), which is currently the most widely used AI language, was developed in 1958. Another widely used AI language is PROLOG (PROgramming in LOGic). Additional languages used in AI system development can be found in section 3.

Although there are other methods of encoding knowledge, ESs generally encode knowledge through a set of rules that are in the form of IF THEN statements. ESs compute a sequence of strings representing the steps in the solution to a problem. To simulate the human reasoning process, ESs apply specific knowledge and inferences. They are able to deal with a wide range of problems, as would a human expert.

High performance of ESs can only be achieved on powerful computing platforms because a great number of rules are needed in the knowledge base.

2.2.1 Expert System Architecture.

The main sections of an ES are a knowledge base, a database, and an inference mechanism. An interface, whether human or with another system, must also be included. Figure 2.2-1 shows the ES architecture. The database contains the current state of the ES as it works toward a solution. The knowledge base contains domain knowledge which is applied to the problem at hand. Information in the knowledge base may be in the form of statements, such as the (attribute) of (object) is (value), or it may be in the form of an IF THEN statement (rule). Attributes are terms such as age, distance, or temperature, and they may not be known precisely (Togai and Watanabe 1992). The inference mechanism, or inference engine, controls the use of the knowledge base and database in solving the problem.

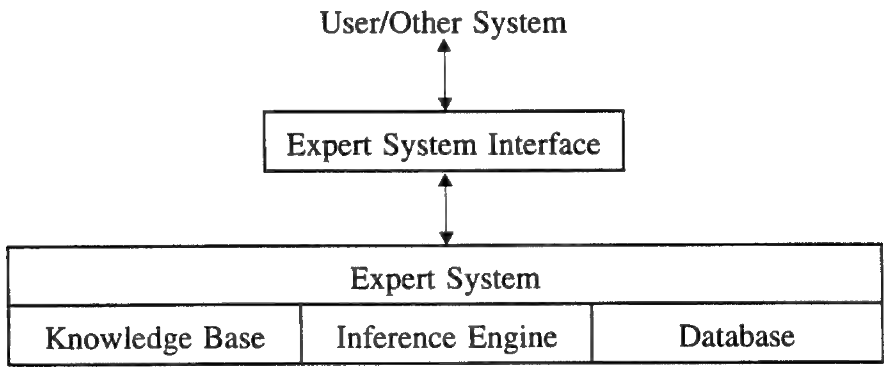


FIGURE 2.2-1. EXPERT SYSTEM ARCHITECTURE

2.2.1.1 Knowledge Base.

The knowledge base contains codified knowledge about a specific domain. It is a collection of facts and heuristics. The power of an ES depends more on the quality of its knowledge base than on the nature of its inference engine. Large knowledge bases can be difficult to develop and maintain. This is a major roadblock in developing ESs that contain large knowledge bases.

2.2.1.2 Database.

The database contains an organized collection of specific data concerned with the problem to be solved. It represents what the computer believes to be either a partial or full solution to the problem. The inference engine acts as a gatekeeper between the database and the program using it. The inference engine is responsible for adding and deleting beliefs and performing certain classes of inference, such as adding facts or handling requests for information (Charniak and McDermott 1985).

2.2.1.3 Inference Engine.

The function of the inference engine is to use knowledge in the knowledge base, along with acquired knowledge about the specific problem, to form an expert solution. To achieve defined goals, the inference engine controls and executes reasoning using the knowledge contained in the database and the knowledge base. The system's general problem solving knowledge is contained in the inference engine.

The inference engine must be able to determine when it is appropriate to ask the user for information and when it should consult the knowledge base and database. The inference engine needs to be flexible to deal with varying situations. It also may be required to handle imprecise and ambiguous information.

A well designed inference engine will be able to make use of many different patterns of evidence and use these to narrow the scope of possible conclusions. Various search techniques are used by the inference engine and applied to the knowledge base and database to reach a solution. Search techniques are examined later in this report.

2.2.2 Expert System Implementation.

ESs that are most accepted by users and are most successful have a number of common qualities (Hall and Kandel 1992):

- Problem domain has limited focus.
- User interaction is minimal.
- Problem domains are well defined and manageable – the more limited and focused the domain, the less knowledge is needed and the less information that must be verified and tracked.
- Experts are available for consultation.
- Solution verification is straightforward and noncontroversial.
- The systems are not consultation systems, i.e., systems that claim to give indisputable expert advice.

ESs that have not gained wide acceptance by users typically do not have a good user interface, are slow in responding, and often ask too many questions of the user. Many unsuccessful systems are consultation or diagnostic ESs that do not explain their problem solving processes to the user's satisfaction.

Rolston (1988) categorizes current ESs into three general classes:

- Assistant – Performs a technically limited subset of an expert's task; however, generally it is an economically worthwhile system.

- Colleague – Performs a significant subset of an expert's task.
- Expert – Performs at an expert's level within a given domain. This type of system requires a large knowledge base and powerful development tools.

Most systems fall into the assistant or colleague class. It is difficult to design a system that is capable of replacing an expert in a complex domain.

2.2.3 Experts.

Experts are people who have mastered specific types of problem solving. ESs seek to capture enough of the expert's knowledge to reach the same conclusions as a human expert. Expertise in a task domain requires substantial knowledge about that domain. The system attempts to generate the heuristic reasoning of the human expert without trying to understand it.

One of the most important capabilities of a human expert is the ability to deal with imprecise, incomplete, and ambiguous information. This is one of the most difficult properties to duplicate. Current ES technology is not capable of dealing with uncertainty as effectively as human experts. This is an area that requires more research.

2.2.4 Explanation of Reasoning.

The ability of an ES to justify to the user exactly how it came to a particular conclusion is important. The user will gain trust in the system more quickly if there is an understanding of how a conclusion was reached. Also, errors, inconsistencies, and omissions are more easily detected in the debugging process if the path of reasoning can be traced.

ESs should have the ability to tailor an explanation to the level of understanding of the user. A person well versed in a particular field can understand a higher level explanation than a novice.

At each step of the process, one can monitor the actions the ES is taking and determine if they are consistent with the objectives of the task.

2.2.5 Knowledge Representation.

Knowledge is the fundamental part of an ES and is where the system achieves its power to solve problems (Rolston 1988). The effective representation of domain knowledge is considered the key to the success of AI programs. When choosing a form of knowledge representation, an important consideration for the designer should be the ease with which knowledge can be changed and updated. Flexibility is essential to handle input from human experts who often may change their minds. It is necessary to model a knowledge domain accurately.

Knowledge representation language is the method used to represent facts. Knowledge is defined in terms that can be manipulated by programs. Two common representation languages are natural language (English language sentences) and symbolic (representations of objects at the knowledge level defined in terms of symbols). Basic criteria for a knowledge representation language are as follows (Fikes and Kehler 1985):

- Expressive power – Experts must be able to communicate their knowledge effectively to the system.
- Understandability – Experts must be able to understand what the system knows.
- Accessibility – The system must be able to use the information it has been given.

The knowledge representation must facilitate inferencing, which converts the explicit set of beliefs about a problem domain to a larger effective set of beliefs. Three main ways of representing knowledge in an ES are rules, semantic networks, and frames.

Rules are a formal method of representing strategies, recommendations, and directives. Years of problem solving experience in a particular domain result in empirical associations that are suitable for representation by use of rules. An inference engine sorts through the rules and available data to reach a conclusion.

Semantic networks originally were developed to be psychological human memory models. Their use is now commonplace in the AI field. The network represents knowledge by using nodes and arcs. Nodes can represent objects, concepts, or events, while arcs are used to represent relationships or hierarchies between the nodes.

Frames are similar to semantic networks in that both nodes and arcs are used for representing knowledge. For frames, however, the nodes can take on a collection of attributes with their associated values. Each attribute can have attached procedures that execute when a value is changed.

2.2.5.1 Rule-Based Systems.

Rules are a subset of predicate calculus with additional components to signify the way information in the rules should be used for reasoning (Fikes and Kehler 1985). Predicate calculus represents knowledge about real-world facts as statements.

Most current AI technology is rule-based. Rule-based systems constitute the best presently available means for codifying the problem solving methods of human experts. They are the most commercially successful AI products.

Rules are the most popular and effective representational form for behavioral knowledge in knowledge systems. Rules are fairly simple to work with and are relatively independent of each other. This independence allows incremental construction of AI systems. Rules also are able to represent different types of knowledge. Experts find it easy to state their knowledge in the form of rules. Rules lend themselves to expressing the heuristic knowledge that comes from experience (Friedland 1985).

Experts generally express most of their problem solving methods in terms of situation-action rules. Rule-based systems can incorporate rules that imitate the way experts reason. Rules must allow one to encode an expert's knowledge as thoroughly and efficiently as possible, and the rule-based system must be able to convey its knowledge and reasoning schemes to people of varying skill levels. Other types of systems lack the reasoning schemes used by experts and, therefore, cannot solve practical problems efficiently.

All rule-based systems have several properties in common (Hayes-Roth 1985):

- They use IF THEN rules to incorporate human knowledge.
- They use larger knowledge bases to perform at a higher skill level.
- They have the ability to solve a wide range of problems by combining rules appropriately.
- They determine the best sequence of rules to pursue.
- They can explain their reasoning by retracing their steps.

The rule base contains an expert's knowledge that is used to solve a problem. A rule-based controller models the behavior of the human expert, as opposed to modeling the problem solving methodology. This is a flexible approach to codifying knowledge and is well-suited for representing the knowledge used to control a system.

The usefulness of a rule-based system is dependent upon the fidelity of knowledge and problem representation. Problem solving using a rule-based system goes through many cycles of identifying the rules that have bearing on a particular problem and then applying those rules to solve the problem or achieve the end goal.

Rules can express the following (Hayes-Roth 1985):

- Deductive knowledge – Logical relationships, inference, verification, and evaluation.
- Goal-oriented knowledge – Used in seeking problem solutions.
- Causal relationships – Used to determine possible causes for specified events.

Control rules represent the expert's problem solving strategy. These rules are a way to achieve control over the system while maintaining clarity. There are two types of control rules: those that represent knowledge and those that represent the problem solving strategy. The rules that represent the problem solving strategy are a special type of ES knowledge called meta-knowledge. Basically, meta-knowledge is knowledge about the system knowledge. However, meta-knowledge often can worsen system transparency by complicating testing and making maintenance difficult. System transparency means that the system's knowledge is dependant neither on its location in the system nor on any of the other knowledge found in the system. Further research is needed so that ways to better represent meta-knowledge can be identified. For applications that are more than trivial, transparency and simplicity of the code seem to diminish (Coats 1991). Separating declarative and procedural knowledge makes the system easier to code, debug, and maintain.

If rules are transparent, their meaning is completely independent of their location in the rule base. However, this means the order in which the rules are processed is difficult to determine by examining the rules themselves. Conventional programming has low transparency but high behavioral visibility. Conventional programming controls the way programs process information. Controlling the way programs process information is detrimental to rule-based programming since it creates knowledge bases that are difficult to read and maintain. ESs are programmed to tell the system what to know, while conventional programming instructs the system what to do.

Sometimes rules may include procedural content. They represent knowledge of procedures and methods to process data. Generally, there is little information in the knowledge base about when rules should fire (execute). In theory, using rules to represent procedural knowledge can be risky because it can cause knowledge to become intertwined. When this occurs, it becomes difficult to follow the steps taken to solve the problem. The rule base and the inference engine which processes the rule base should be separate. Global control strategies should be context free. The selected control strategies may be applied regardless of the situation.

When procedural content is included in the rules, they become less independent of each other and their position begins to play a crucial role in producing correct results. Proceduralizing among the rules, as opposed to within the rules, also should be avoided. Rules should not be grouped together to imply procedures as a group. Rules should reflect knowledge, not procedures, and avoid implicit or hidden meanings. A particular problem may be more suited to traditional programming techniques when procedural content within the rules cannot be avoided.

When debugging a conventional program, normally the challenge for the programmer is in determining which statements are out of order. The sequence of events is followed easily. In an ES, the challenge is to determine how the inference engine is processing the rules, ensuring that the rules are correct, and adjusting the inference engine's method of processing the rules.

Conventional computer programming requires programmers to think procedurally, while rule-based programming requires programmers to think more analytically. Rule programming is similar to conventional programming because it requires mental modeling of state changes, syntactic and semantic checking of rule conditions, and heuristic methods for validating and verifying a proposed system (Hayes-Roth 1985).

As with conventional computer programming, structure is critical. A well-structured rule base is easier to test and maintain and better represents knowledge. In classic rule-based ESs, an inference engine processes a knowledge base composed of rules. The separation of the knowledge base and the processor of the knowledge makes the rule-based ES easier to develop, test, and maintain. Because of this, rules can be manipulated without affecting processing or other rules.

Automating expertise in specialized tasks generally requires a few hundred to a few thousand heuristic rules (Hayes-Roth 1985).

A rule-based system consists of three elements:

- A working memory (database) – Contains data structures that represent the current solution state of the system or intermediate results. Contents change until the final solution is found.

- A set of rules (knowledge base) – Contains IF THEN statements from rules and facts. This is a permanent, static structure.

- An inference engine – Contains hardware and software that matches the conditions in the rules to the entries in working memory and fires the rules if there is a match.

Two useful characteristic features of rule-based systems are incremental development, which allows knowledge to be refined and added, and explanation of reasoning, which aids in understanding computer systems.

Guidelines for a well-structured knowledge base are given below (Pederson 1991):

- Keep conclusions simple – Rules should only update one attribute at a time; the results are clear and easy to follow; and it is easy to tell which rule is responsible for which action.

- Keep rules free of procedural content.

- Minimize the use of ELSE constructs.

A poorly structured rule-based ES would be analogous to excessive use of GOTO statements in conventional programming. It is difficult to follow the path that the program is taking, and, therefore, more difficult to debug. Rules with multiple conclusions or the use of ELSE statements result in a process that is confusing to follow.

For rule-based ESs to be successful, they should have the following characteristics (Weiss and Kulikowski 1984):

- The conclusions must be generated by the system from a finite set of discrete and prespecifiable elements.

- The evidence about the problem must be obtained reliably by the user of the system or the system itself.

- The initial assumptions must limit the problem to a highly specialized area.

- A knowledge base must be available to link evidence about the problem to conclusions.

- A reasoning control strategy must be designed to guide the reasoning of the system and make its output correspond to that of a human expert.

The types of knowledge that can be included in rule-based systems follow (Hayes-Roth 1985):

- Inferences that follow from observations
- Generalizations of data
- Conditions for achieving a goal
- Best places to find relevant information
- Best strategies for eliminating uncertainty and minimizing risks
- Probable causes of symptoms
- Likely outcomes of hypothetical situations

Future goals for rule-based systems include the following:

- Increasing the size of the rule base to 10,000 or more rules

- Increasing the processing speed
- Using more inference techniques
- Improving reasoning with uncertainty
- Simplifying the process of creating and extending knowledge bases
- Sharing knowledge bases
- Improving architectures
- Generating more efficient ES development tools

Rule-based systems have the ability to capture, represent, store, distribute, reason with, and apply human knowledge. They are a practical solution to building automated experts and are especially applicable in tasks that require consistency and practical experience. Although AI developers have generated other approaches, the rule-based method is the only one that consistently produces accurate results. Rule-based systems offer advantages that cannot be found in traditional programming techniques (Hayes-Roth 1985):

- Modularity

- Knowledge bases that store rules and facts and make decisions

- Explanations of results

- Easily understood beliefs and problem solving techniques

- Inference chains (the path of reasoning and searching that the inference engine follows to achieve a goal) that are efficient

<u>2.2.5.1.1 Rule Architecture</u>.

Rules basically are independent pieces of knowledge. As the rule base gets larger, rule components (antecedent, conclusion) must be flexible and independent enough so that extension and maintenance of the knowledge base will not adversely affect them (Hayes-Roth 1985). The rules are in the form of IF THEN statements and may contain the knowledge of one or more experts. The inference engine manages the rule base by using search expressions or algorithms.

The left side of the rule is the situation recognition part, known as the antecedent or premise. The right side is the action part, referred to as the conclusion or consequent. The antecedent expresses some condition in the state of the database and describes if and when it is satisfied. Usually it consists of a sequence of clauses connected by AND or OR, which serve as minimum and maximum operators. The action part specifies what changes are to be made to the database when a rule is satisfied. The inference engine monitors the facts in the database and when an antecedent is satisfied, it executes the corresponding action.

The current facts, along with the inference engine, make up the computing environment. Together they serve to interpret the current state, understand the meaning of the rules, and apply the rules appropriately.

When the consequent defines an action, scheduling the action for execution satisfies the antecedent. When the consequent defines a conclusion, inferring the conclusion will satisfy the antecedent.

There are several functions that the rules perform (Hayes-Roth 1985):

- Simplify auditing and explanation by having the ability to trace every result to its antecedent and intermediate inferences.
- Simulate reasoning by expressing logical relationships.
- Simulate human understanding.
- Simulate human decision making by using conditional rules to express heuristics.

The rules must cover all possible combinations. If no antecedents are satisfied, then no action will be taken.

2.2.5.1.2 Reasoning with Inexact Concepts.

Certainty Factors (CFs) are values that represent the level of belief associated with a fact or a rule. They are a means of providing a judgement about the certainty of a conclusion, which is determined by combining degrees of belief and disbelief. The CF is based on the level of certainty known about conditions in the premise. Every time a new rule is considered, a new CF is calculated. The use of CFs is a technique for supplementing existing reasoning processes with information regarding uncertainty. Normally, an interval between 0 and 1 is used to represent the CF, with 1 being absolute certainty.

2.2.5.1.3 Designing a Rule-Based Expert System.

The main steps that must be taken to develop a rule-based system include the following:

- Define the task – Determine the problems the rule-based system will be used to solve.
- Determine the available resources – Include expertise, computing facilities, and funds.
- Code the knowledge and form the rules.
- Build a prototype – Create the rule base, implement the method of inference, experiment with the system, analyze the important issues, and generalize the rules.
- Validate and test the system – Eliminate bugs and inconsistencies.

The construction process of an ES consists of several stages and each stage must be tested and revised a number of times.

2.2.5.2 Semantic Networks.

A semantic or associative network is a way of representing knowledge. Nodes represent objects, concepts, or situations in the domain. Arcs (links) represent relationships between the nodes. A network is a representation of the relations between elements in a domain using interweaving and crisscrossing arcs (Rolston 1988). A semantic network is a graphical representation of a network. Figure 2.2-2 shows the semantic network architecture. Arcs can also have weights, which indicate the strength of the relationship between the nodes.

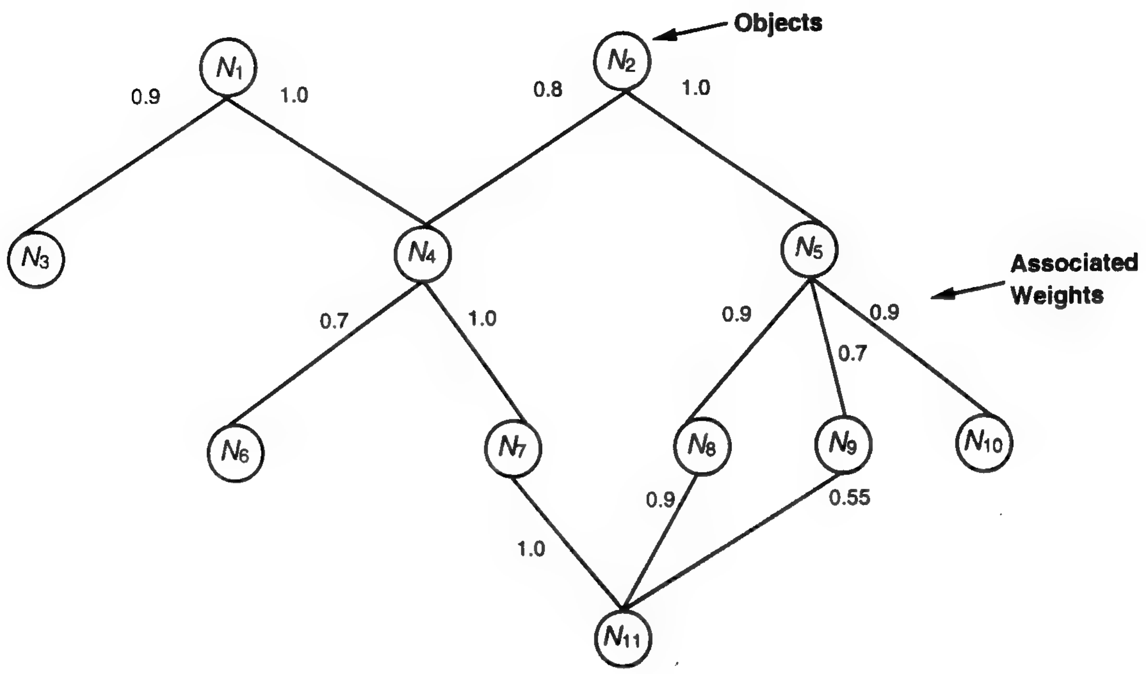


FIGURE 2.2-2. SEMANTIC NETWORK STRUCTURE

Arc labels indicate the basis of the relationship between the two arcs. An arc can be viewed as something that is asserted to be true about one element relative to another (Rolston 1988). For example, "DC-10" is linked to "aircraft."

Semantic networks are easy to read. They are based on an object-oriented illustration, meaning the representation is based on objects that are treated as independent pieces of knowledge in the computer. Semantic networks are represented in a computer by using memory addresses to depict the arcs that go from one node to another. This facilitates access to all parts of the semantic network.

A drawback of using semantic network representation is that only facts can be represented easily. If logical connectives are attempted, the structure of the network may become very complicated.

2.2.5.3 Frames.

The organization of frames is similar to that of semantic networks. Frames consist of a set of slots that contain data, procedures, or pointers to other frames. The pointers can create nested frames. When slot relations warrant some action, the procedures will define the type of action to be taken (Hall and Kandel 1992). Frames provide structured representation of an object or class of objects. Table 2.2-1 illustrates the frame architecture. Frames are well-suited for applications in which complex descriptions are needed to depict the domain accurately. They provide the knowledge base builder with a means of describing the types of specialized objects that the system must model.

TABLE 2.2-1. FRAME STRUCTURE

Frame Label	Facette 1	Facette 2	Facette 3
$slot_1$	$value_{11}$	$value_{12}$	$value_{13}$
$slot_2$	$value_{21}$	$value_{22}$	$value_{23}$
. . .	. . .	. . .	. . .
. . .	. . .	. . .	. . .
$slot_n$	$value_{n1}$	$value_{n2}$	$value_{n3}$

Frames provide the ability to interpret new situations on the basis of knowledge gained from similar situations. Often, in the beginning of frame development, a best fit frame must be chosen for the current situation because there may not be an exact frame that fits at that time. As development continues, frames may be added and/or changed.

Frames also can be used to represent rules. This provides a method of grouping the rules into classes. Frame-based representation can ease the rule management task significantly. When systems become very large, it is difficult for system designers to follow the interactions between rules, debug them, and control their behavior. Frames provide a means of organizing and indexing collections of rules according to their intended use.

Frames are appropriate for representing prototypical knowledge and the properties of objects to be modeled. Frames identify specific pieces of knowledge that belong together in a group. Frames are made up of pieces of knowledge that are typical characteristics of the ideas associated with that particular frame label. The pieces of knowledge in the frame are called facettes and they take on values that uniquely identify the characteristics of a specific object.

For example, a particular frame may be labeled Aircraft Accident. The facettes that may be contained in that frame include components normally associated with any aircraft accident:

- Cause
- Number of passengers
- Number of crew members
- Number of injuries
- Location
- Date
- Time

For an aircraft accident, these facettes are filled with values that can identify a specific accident. When the values are entered into the facette, the result is a particular instance of the general concept Aircraft Accidents represented as a frame (Adeli 1990).

Frames are an excellent way for a knowledge system to organize and direct its reasoning activities. As with semantic networks, frames are an object-oriented form of representation. The use of frames, along with rules, may be a promising area for improvements in knowledge representation. When representing knowledge, rules and frames integrated in a single unified representation scheme work best.

2.2.6 Models.

A model focuses on certain aspects of an occurrence. Each class of models in knowledge engineering selectively emphasizes a different aspect of human expertise and provides alternatives for engineers. There are four basic classes that model human expertise in conjunction with heuristics (Adeli 1990):

- Deep model – Uses a form of heuristics relying on fundamental principles of the field to solve complicated problems. It is a way of relying on the basic, elementary foundations of the domain to determine solutions.
- Implicit model – Uses the expert's intuition to represent knowledge that is experience-based and cannot be expressed clearly in language form.
- Competence model – Uses an experts's domain knowledge to solve problems effectively. Operative knowledge obtained from interaction or operation with a domain system is used, as opposed to specific mental operations.
- Distributed model – Uses the expertise of multiple experts. The experts must interact cooperatively.

The form of the knowledge, such as rules or semantic networks, is the symbolic or representational level of knowledge engineering. The models generate actions and specify the operations the symbolic systems should perform.

2.2.7 Updating the Knowledge Base.

Machine learning is a process by which a system generates and updates its knowledge base. The system learns from past experience. This is the ultimate goal for updating a knowledge base. This area, however, has not been perfected and requires more research.

Inductive and deductive learning are two main learning techniques used in machine learning. Both methods involve using the knowledge base to generate other pieces of information not explicitly expressed in the knowledge base (Adeli 1990). Inductive learning is learning by information repetition. Deductive learning is learning by logical inference: the process of inferring while maintaining the truth of the data that are already stored.

Knowledge also can be updated manually. This process is performed by a knowledge engineer who interprets and encodes knowledge from a human expert. Another form of updating knowledge is for the expert to enter knowledge directly into the system with the assistance of a tool designed to perform the knowledge engineer's tasks. The knowledge engineer is eliminated.

2.2.8 Chaining.

Chaining refers to the path of rules that the inference engine fires to achieve a solution. The inference engine can process information in one of two ways: forward or backward chaining. Some ESs, such as blackboard systems, use a combination of the two methods.

Elements to be considered when choosing a direction for the search include the following (Adeli 1990):

- The number of initial facts versus the number of acceptable conclusions – Search direction should be toward the larger set.
- The average number of choices in each direction – Reasoning should be directed so that the number of possibilities in each cycle is minimized.
- The need for interactive processing – User supplied information is more easily utilized in backward chaining.
- The need for justification of results – Backward chaining naturally lends itself to showing support for a result.

2.2.8.1 Forward Chaining.

Forward chaining uses bottom up or event driven reasoning. A rule is triggered when changes in working memory data produce a situation that matches its antecedent component. This method begins with some initial data and moves down the inference chain until it reaches a solution to the problem. Working memory is used to store the new facts. Working memory also stores initial observations, findings, and conclusions.

The system does not start with any particular goals or subgroup of production rules. It starts with some evidence and proceeds to fire the rules until no more can be fired and the goal has been achieved (Weiss and Kulikowski 1984).

Figure 2.2-3 demonstrates forward chaining. What is currently known (facts) are examined by the rules. Facts are stored in the database, while rules form the knowledge base. There are three rules in this example. The first rule, F&B---> Z, states that if both F and B exist, then Z also exists. The second rule is similar, but uses different facts. Rule three states that if A exists, then D also exists.

In the first pass through the rules, the only fact found true is A. This causes D to be added to the existing facts. Likewise, the second and third passes through the rules find F and then Z to exist. The inference chain which explains the conclusion reached by the system is:

1. Rule A executes and adds D
2. Rules C and D execute and add F
3. Rules F and B execute and add Z

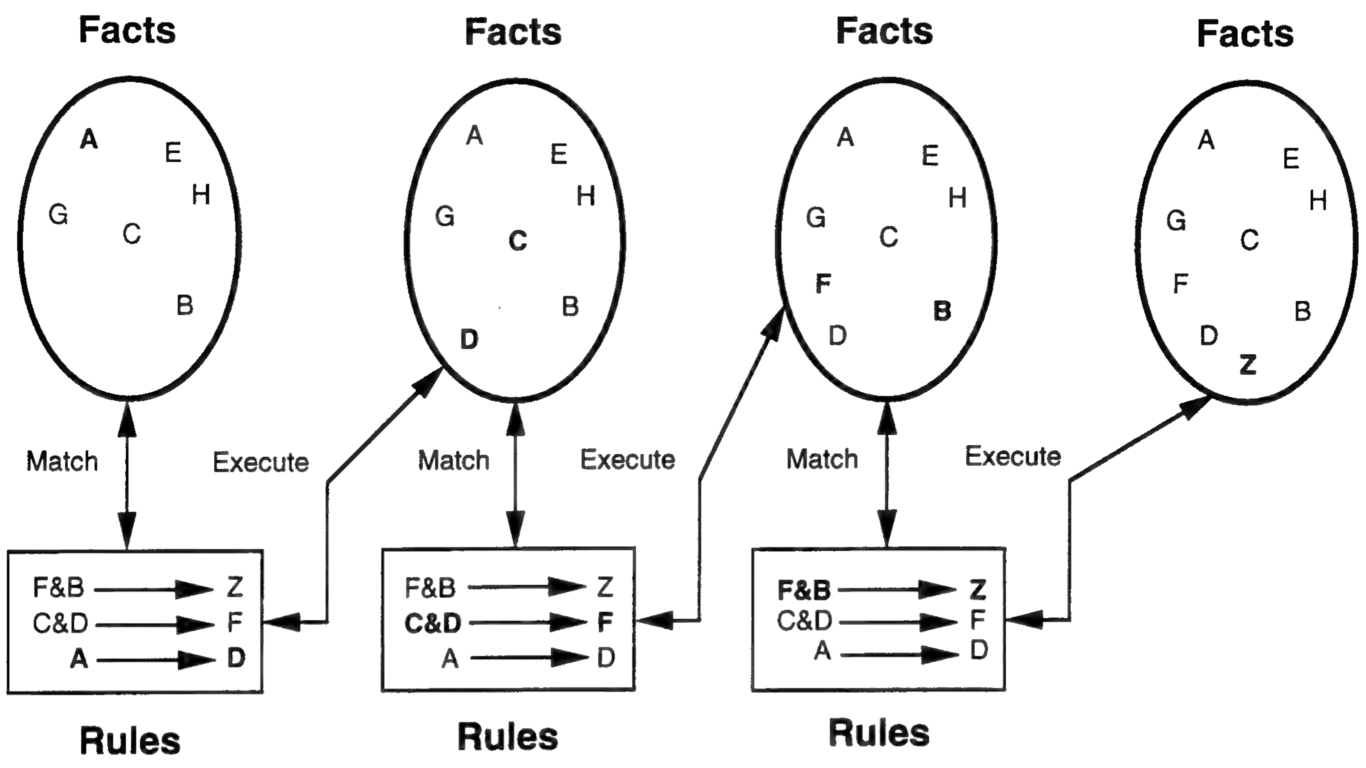


FIGURE 2.2-3. FORWARD CHAINING
(Waterman 1986)

An advantage of forward chaining is that it is fairly simple and easy to understand. However, it may perform some computations that are useless in attaining a certain goal because it may follow a path of firing rules that do not attain a desired goal, or any goal at all (Adeli 1990).

2.2.8.2 Backward Chaining.

Backward chaining is the process of working backward from a known conclusion to find a path of reasoning to justify the conclusion. The goal is assumed to be true and the inference engine searches for evidence to support that conclusion. The rule-based system begins with a goal and successively examines any rules with matching consequent components. The unmet conditions of the antecedent are taken from each applicable rule and then defined as new goals. In backward chaining, working memory is used to note unresolved subgoals. This is also known as goal directed, or top down, reasoning. Backward chaining avoids unnecessary searches; however, it may create contradictory situations for the user by asking questions that are unrelated or out of sequence.

Figure 2.2-4 demonstrates backward chaining. The same three rules and six facts exist in the beginning. Here, the system checks to see if Z is a valid conclusion. During the first pass through the rule base, it is noted that Z is not a current fact and that both F and B are required for Z to exist. Next, it is noted, that for F to exist, C and D must also exist. The third pass indicates that C exists, while passes 4, 5, and 6 determine that D now belongs in the list of current facts. Step 7 then determines that F exists, while step 8 reaches the conclusion that Z exists.

The inference chain which explains the conclusion reached by the system is:

1. Z does not exist, but requires F and B
2. F does not exist, but requires C and D
3. C exists
4. D does not exist, but requires A
5. A exists
6. Rule A executes and adds D
7. Rules C and D execute and add F
8. Rules F and B execute and add Z

2.2.9 Knowledge Acquisition.

Knowledge acquisition is a lengthy process in which the knowledge engineer must meet with a known expert who is willing to provide information. It is a tedious process and creates a bottleneck in ES development. It is one of the most difficult phases of ES building. The knowledge acquisition process is not well understood or well defined (Rolston 1988).

Knowledge can be represented as procedural or declarative. Most declarative schemes are knowledge represented as static facts, along with a limited amount of information that describes how the information should be used.

An ES shell is an ES with an empty knowledge base. ES shells require knowledge engineering that extracts a collection of highly detailed facts from a human expert and programs them into a database. This process requires repeated testing of the program to verify and account for all possible situations.

Both procedural and declarative knowledge are needed for every ES. However, they should be kept as separate as possible. Generally, procedural knowledge is found in the inference engine and declarative knowledge is found in the rule base.

The inference engine utilizes meta-knowledge. Meta-knowledge includes information on how to utilize available knowledge, in what order the knowledge should be used, where the knowledge can be obtained, and the point at which processing is complete.

Knowledge engineering can be an expensive, time consuming, and difficult task for the following reasons (Adeli 1990):

- Vocabulary – The knowledge engineer must comprehend the basic vocabulary of the domain.

- Completeness – The knowledge engineer must understand the domain sufficiently to identify pieces missing from the knowledge base.

- Integration – New information must be added to the knowledge base in a way that will not adversely affect existing knowledge.

- Analysis – The knowledge engineer must have the ability to evaluate the expert's methods for arriving at conclusions.

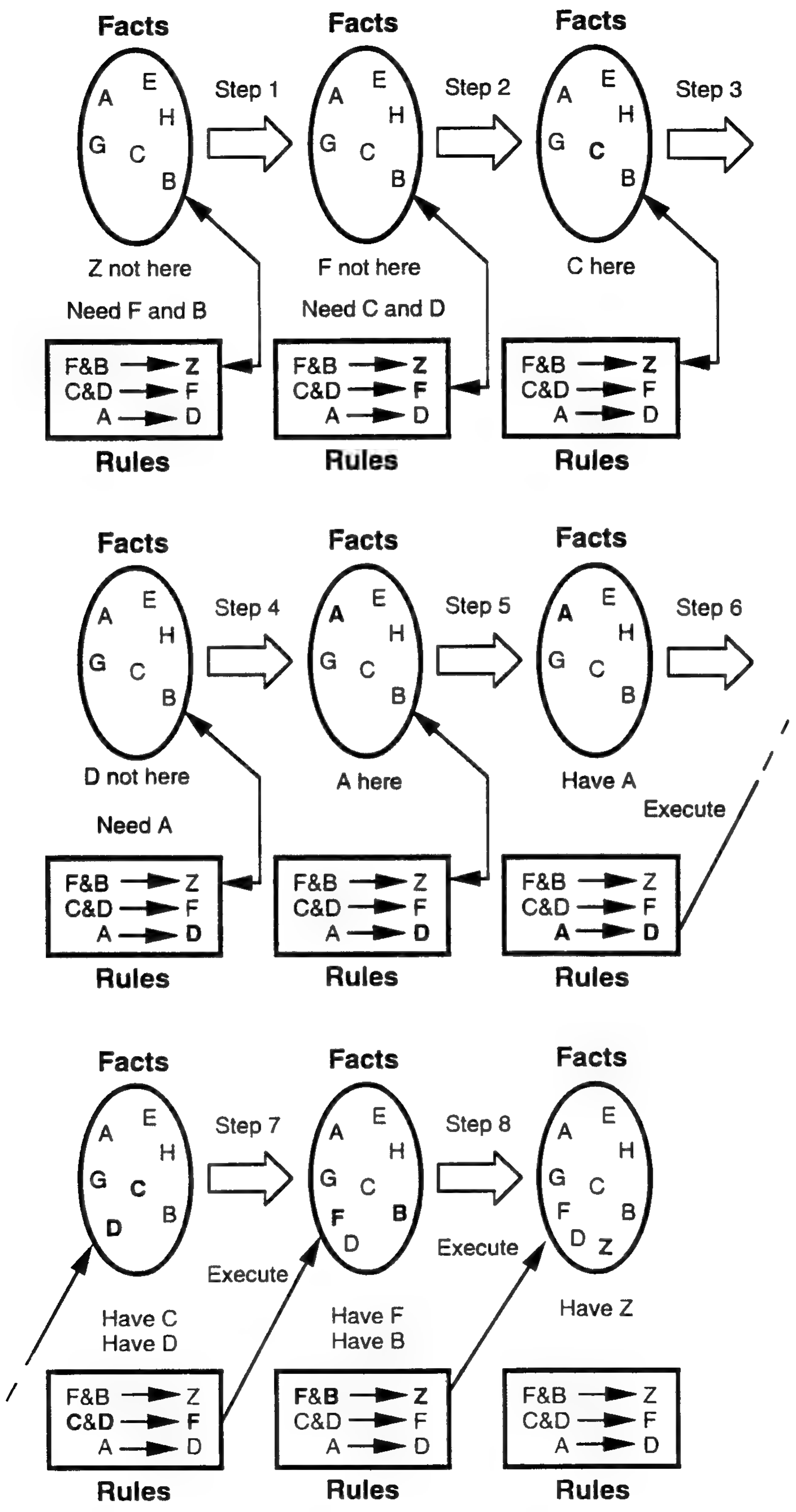


FIGURE 2.2-4. BACKWARD CHAINING
(Waterman 1986)

- Transparency – The knowledge engineer must incorporate the expert's problem solving behavior into the knowledge base.

Knowledge engineers need to have a working knowledge of the domain that is being modeled. They need to become proficient in the field to be able to comprehend the expert's information and to determine how to handle situations. For example, the knowledge engineer must determine how to handle contradictions and inconsistencies between experts. The knowledge engineer may need to resolve conflicting, redundant, subsuming, or missing blocks of knowledge.

In general, the knowledge acquisition process is a transfer of knowledge from existing sources, rather than learning from experience. This may be more difficult than it seems. An expert may have excellent ability to perform tasks, but may not be able to communicate the steps taken to do so.

Several tools have been developed to assist knowledge engineers with prototype development. Some tools may replace the knowledge engineer with an automated acquisition system. Any system intended to replace the knowledge engineer must be able to perform all knowledge engineering tasks.

Following are general requirements for a knowledge acquisition tool (Adeli 1990):

- Ability to be used by experts without outside assistance.
- Ability to access several types of knowledge sources.
- Ability to account for different perspectives (e.g., partial or contradictory input).
- Ability to encompass different forms of knowledge and their relationships.
- Application of knowledge in a variety of domains (domain independence).
- Expression of the knowledge as an algorithm.
- Preparation of the knowledge base for validation.
- Theoretical basis for knowledge acquisition and representation.
- Ability to converge knowledge acquisition into an integrated system.

2.2.10 Applications.

Applications suitable for ESs are presented below:

- Applications where a proven expert is available.

- Systems that entail robot activities.

- Limited domains with well defined expertise.

- Systems that do not require extensive user interaction.

- Problems that do not have a large amount of uncertain information. (An ES will no longer be appropriate when a specific level of uncertainty is reached.)

There are some factors that justify the use of an ES:

- Human experts are rare or soon will be unavailable.

- Identical expertise is needed in many locations.
- The expertise is needed in a hostile environment.

If the problem to be solved can be described by direct definitions and algorithms, it may be solved better by traditional software methods. In addition, if the problem is extremely difficult to define or requires intense judgement, it may be too complex for an ES and will need to be solved by traditional methods (Rolston 1988).

Although problems best solved by ESs are not yet well defined, judgement can be made on the types of problems where ESs can be ruled out:

- The problem is not well defined.
- There are no true experts who have mastered the task.
- Conclusions are based on factors that are not fully understood.
- Conclusions are based on factors that are unpredictable.
- The knowledge required is not stable.
- The risk of making an incorrect decision is high.
- Common sense is required.
- Conventional programming is solving the problem adequately.
- The problem involves learning.

A number of characteristics are common to ESs in use today:

- There are proven experts available.
- The task requires a well defined domain without changing rules.
- The problem requires more cognition than physical attributes.
- The problem takes an expert a few minutes to a few hours to solve.
- The experts teach their skills and are able to explain their reasoning.
- Solving the problem will produce considerable benefits.
- The problem solution requires no common sense.
- There is low risk if a bad decision is made.

An ES attempts to find an acceptable solution, but not necessarily the best solution, in a reasonable amount of time. It deals with a specific problem domain, as opposed to attempting to mimic human behavior in all domains (Rolston 1988).

2.2.11 Designing a System.

The task of building an ES is comprised mainly of teaching the system enough facts and heuristics to enable it to perform competently in a particular problem solving context.

The early stages of system design are the most difficult. It is difficult to establish all specifications at the beginning of the design phase. The designer should concentrate on developing a small prototype first and then make major revisions to it. The prototype model starts simply and becomes increasingly complex, until the system performs at an acceptable level. The prototype will provide direction and focus for the system and also give the expert a basis for suggestions.

Final specifications are difficult to define at the start because the environment is dynamic. Even though experts are knowledgeable in their fields, they may not know how to formalize the reasoning that is used to arrive at a solution. Therefore, there will be many changes made during the knowledge acquisition process before an expert and a designer are satisfied with the results. Once the first prototype is built, the knowledge acquisition process will proceed more quickly. At this point there is a more limited and focused scope. Without a prototype, the designer may be overwhelmed with a mass of abstract suggestions and knowledge that will need to be organized and focused in some fashion.

Some other considerations for designing an ES include selecting a realistic application and the usefulness of the final system. The knowledge engineer must ensure that knowledge is properly extracted and coded for computer representation.

To test ES accuracy, the expert is asked to verify the results. If changes are made to the knowledge base, the effect on all knowledge must be considered.

A particular problem is characterized by an initial state and a goal state description. The state of the system identifies the values of the dynamic parameters that determine problem solutions. The initial state description tells the planning system the current system status. The goal state description tells the planning system what the goal system status should be once the plan has been executed.

Most ESs use a hierarchial goal structure which subdivides each goal into subgoals which are then further divided. Solutions can be built incrementally as each subgoal is achieved.

2.2.12 Heuristic Reasoning.

An heuristic is any rule of thumb or strategy that is used to limit the time required to search for solutions in large problem areas (Coats 1991). Heuristics is the study of the processes involved in solving a problem. Heuristic reasoning is reasoning in the form of general rules, hunches, or rules of thumb that are approximate and generally have been acquired through years of experience. Heuristics contribute to an ES's power and flexibility and set them apart from traditional software.

Heuristic reasoning can provide direction to a search process, thereby reducing the area in the knowledge base that must be searched. It can assist in processes such as eliminating entire branches of a search tree, selecting a general path to follow, and selecting the next node to expand.

Heuristic reasoning is used in the absence of more precise control mechanisms because it uses rules of thumb that are meant to produce acceptable results in most cases. A drawback of heuristic reasoning is that it could lead a search down the wrong path.

The technique of programming used by most ES programmers is referred to as heuristic programming. The exact behavior of the system may not be precisely specified or predictable by the programmer (Denning 1986). These are techniques that can only be acquired from years of experience in a specialist domain. This type of knowledge tends to be inexact, uncertain, and incomplete. Most of the time, heuristics offer reasonable but not necessarily the best solutions.

2.2.13 Searching.

Occasionally, more than one rule may be eligible to fire at the same time. A methodology to resolve these conflicts must be established. The most systematic ways of resolving conflicts are breadth first and depth first searching. They can be used with either forward or backward chaining (Adeli 1990). Both of these search techniques can be used with rule-based systems.

2.2.13.1 Breadth First.

In a breadth first search, all possible subgoals on one level are considered before the next level is examined (Adeli 1990). Increasingly broad segments of the solution space are created and each level is checked for a goal state. All possible successor nodes are created from each state by applying applicable operators to the existing nodes. This ensures that the nodes at a given level are checked before the next level is addressed. Figure 2.2-5 shows the breadth first search strategy.

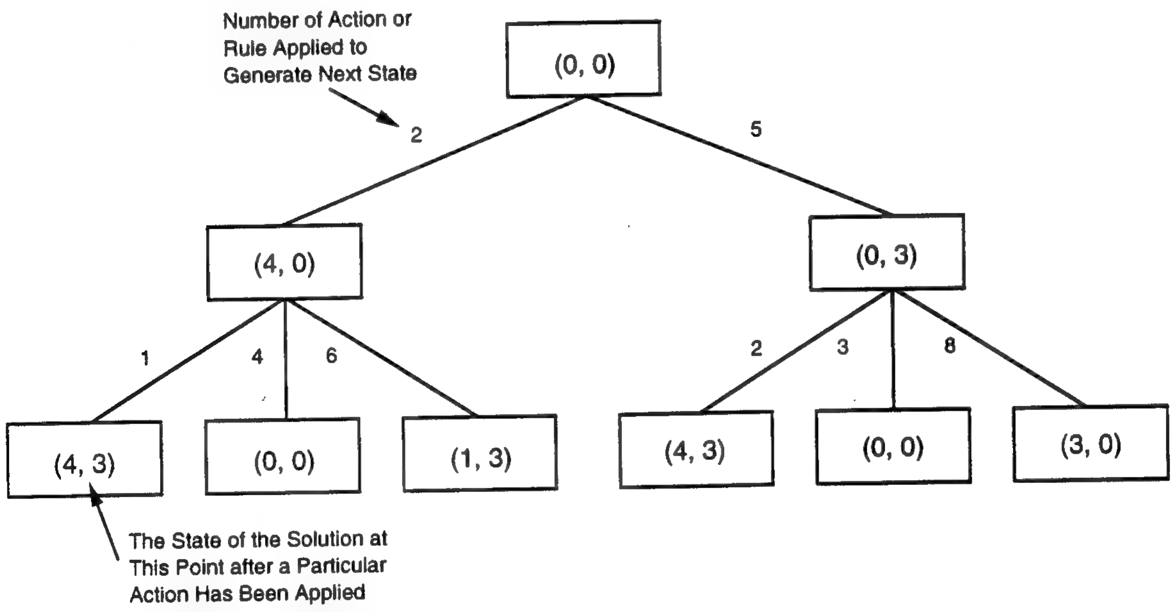

Number of Action or Rule Applied to Generate Next State
(0, 0)
2
5
(4, 0)
(0, 3)
1
4
6
2
3
8
(4, 3)
(0, 0)
(1, 3)
(4, 3)
(0, 0)
(3, 0)
The State of the Solution at This Point after a Particular Action Has Been Applied

FIGURE 2.2-5. BREADTH FIRST SEARCH TREE
(Rich and Knight 1991)

Theoretically, this strategy will find a solution, if there is one, and will find the shortest path to that solution. However, each time the system moves to a new level, the number of nodes grows and so does the time required for the search. The ability to come to a conclusion given unlimited time is not considered expert behavior. It is a good search direction for systems that have access to large volumes of time and memory (Rolston 1988).

2.2.13.2 Depth First.

Depth first searching entails pursuing each branch of a tree completely before considering another branch. When a goal or a subgoal matches the left side of a rule, all premises of that rule are examined before another subgoal is considered. This technique takes one idea and pursues it until a goal is reached or the limit of that branch of the tree is reached (Adeli 1990). The choice of the node used in continuing the path may be based on random selection.

This method does not require as much memory as the breadth first search and it is a faster choice for solutions that may be far down in the search tree. However, a particular solution path may not be the shortest available, and a particular search path may be stopped too soon for fear of an infinite path. Figure 2.2-6 shows the depth first search strategy.

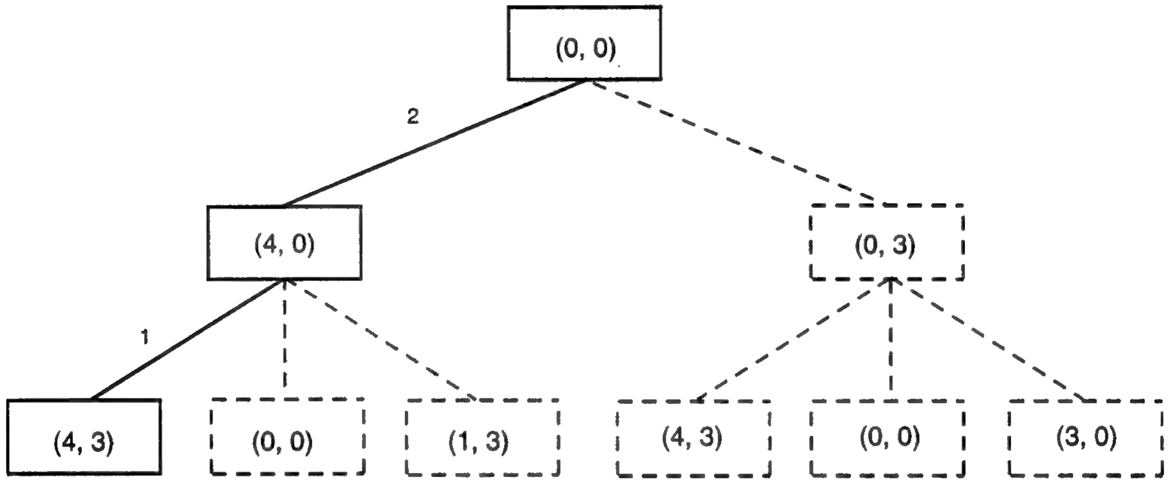


FIGURE 2.2-6. DEPTH FIRST SEARCH TREE
(Rich and Knight 1991)

2.2.14 Expert System Drawbacks.

The purpose of an ES is to perform a difficult task or resolve a substantial problem. It is rare for an ES to outperform a human expert and, if it does, it is because the ES forgets less rather than remembers more. Problems that are suited for ESs may require long development and testing times and can be costly. Even then, the desired results may not occur.

A number of problems exist with ES methodology, code, knowledge acquisition, and validation (Coats 1991):

- Methodology – The use of IF THEN statements can be clumsy and unsuitable to express knowledge effectively.
- Code – Can be difficult to understand, debug, or maintain, and does not always provide adequate user interface.
- Knowledge acquisition – It is not always clear how to resolve conflicting views among acknowledged experts.

- Validation – Methods for validation are still being researched. Perhaps the basic question should be "does the user of the ES make better decisions by using it?"

One main concern about ESs is reliability. Their effectiveness ultimately relies on the system developers. Very large databases are hard to maintain and it is difficult to account for all possible situations. Table 2.2-2 presents potential drawbacks of ESs.

TABLE 2.2-2. POTENTIAL EXPERT SYSTEM DRAWBACKS

Knowledge Base
• Difficult to test
• Difficult to maintain and update knowledge
• Difficult to keep separated from inference engine
• No cognition/learning capabilities to create new rules
• Difficult to achieve good structure
• Difficult to solve dynamic problems
Human Error
• Knowledge engineer may miss gaps in the knowledge base
• Knowledge engineer must ask correct questions
• Expert must relay problem solving information efficiently
• Designers must perform well for ES to perform well
Rules
• Processing hardware not yet developed
• Development can be costly
• Reliability of heuristic rules versus precise specifications used in conventional programming to solve problems

Unless ESs are designed to provide reliability and maintainability, they will not be capable of dealing with complex, real life situations. They need to be adaptable to handle situations that lack complete information. Care must be taken to determine whether an ES is suited to the problem.

There are many influences in an ES domain that may introduce uncertainty into a problem (Hall and Kandel 1992):

- Characteristics of the solution to the problem
- Questions asked to determine the solution
- Knowledge acquisition process
- Reasoning process of an expert
- Knowledge representation language

One way of handling the imprecision would be to introduce fuzzy reasoning techniques into an ES.

Currently feasible systems may have the following limitations (Rolston 1988):

- The knowledge is acquired from a small number of human experts.
- The application is limited to a specific domain.
- There is limited ability to handle time and space reasoning.
- The task cannot rely on a large body of general or common sense knowledge.
- The knowledge to perform the task must be reasonable, complete, correct, and stable.

2.2.15 Future of Expert Systems.

Currently, interest in ESs is high and their applications are increasing. Relatively low cost computers are being developed with large memories, ample disk space, and faster processing capabilities. These capabilities will enhance the possibilities for developing ESs with broader applications.

ESs are most accepted if they critique conclusions or act as advisors. Some workers feel threatened by ESs, fearing that such a system may replace them. This, however, is not the case. ESs will help people by acting as a source of information and an advisor. The goal of using an ES is to improve productivity and the quality of decision making.

In the future, powerful software and hardware may be all that is necessary to develop a system. The ultimate goal will be to eliminate the role of the knowledge engineer and enable the experts to encode their own knowledge directly on the computer. Attention should be focused on methods of extracting and representing knowledge.

2.2.16 Blackboard Systems.

A blackboard system is a type of ES with a slightly different architecture. It is a multiple knowledge source system in which data are shared between sources. A knowledge source system is a program whose knowledge is in a separate database that can be manipulated by the user. The shared data provide communication and cooperation between the experts. The rules have access to all parts of the blackboard. The blackboard serves as the input to the rules and as a record of the outcome of the rules. The blackboard can display the best interpretation of a specific problem at any given time.

Blackboard systems are well suited for applications that use diverse or uncertain data. Blackboard systems use opportunistic reasoning. They deal with problems in ways that are not uniform or predictable.

The blackboard is the controller for the entire structure. It records and organizes solutions that are generated during the problem solving process. Knowledge bases can communicate only through the blackboard and they can generate input to the incremental solution on the blackboard whenever they have relevant information. There is no priority or hierarchy established for knowledge bases to contribute to the blackboard, and the system is not committed to forward or backward chaining. It can implement whichever it sees fit.

The objective of each knowledge source is to contribute information that will lead to the solution of the problem. Each knowledge source must recognize when conditions are appropriate for its contribution to the solution. Each knowledge source has a separate inference engine. Some architectures contain a scheduler, which chooses the next knowledge source to contribute data.

Blackboard systems solve problems by dividing them into several smaller subproblems to be solved separately. The problem dividing method influences how well the problem is solved.

Advantages of a blackboard system include the following (Engelmore and Morgan 1988):

- Many knowledge sources participate in forming a solution.
- Each knowledge source has continual access to the current state of the solution and can contribute when needed.
- The solution is built incrementally.
- Both forward and backward chaining can be used.

A blackboard can be constructed in several different ways. It may have one or more blackboard panels corresponding to different solution perspectives. It may have global or sub-blackboards that are only accessible to certain knowledge bases. It can also have competing blackboard entries in which the best will be chosen.

The classic blackboard system architecture, as shown in figure 2.2-7, consists of the following (Cohen 1985):

- Several knowledge sources – Collections of production rules
- A database or blackboard
- A scheduler (optional) – An independent knowledge source that determines which knowledge source should be activated next to determine a solution.

2.2.16.1 Knowledge Sources.

Each knowledge source is an ES in its own right, with an internal structure which can be independent from other knowledge sources. For example, the structure can be rule- or procedure-based, or can be homogeneous or heterogeneous. The form of the rules and the way knowledge is represented can vary from source to source.

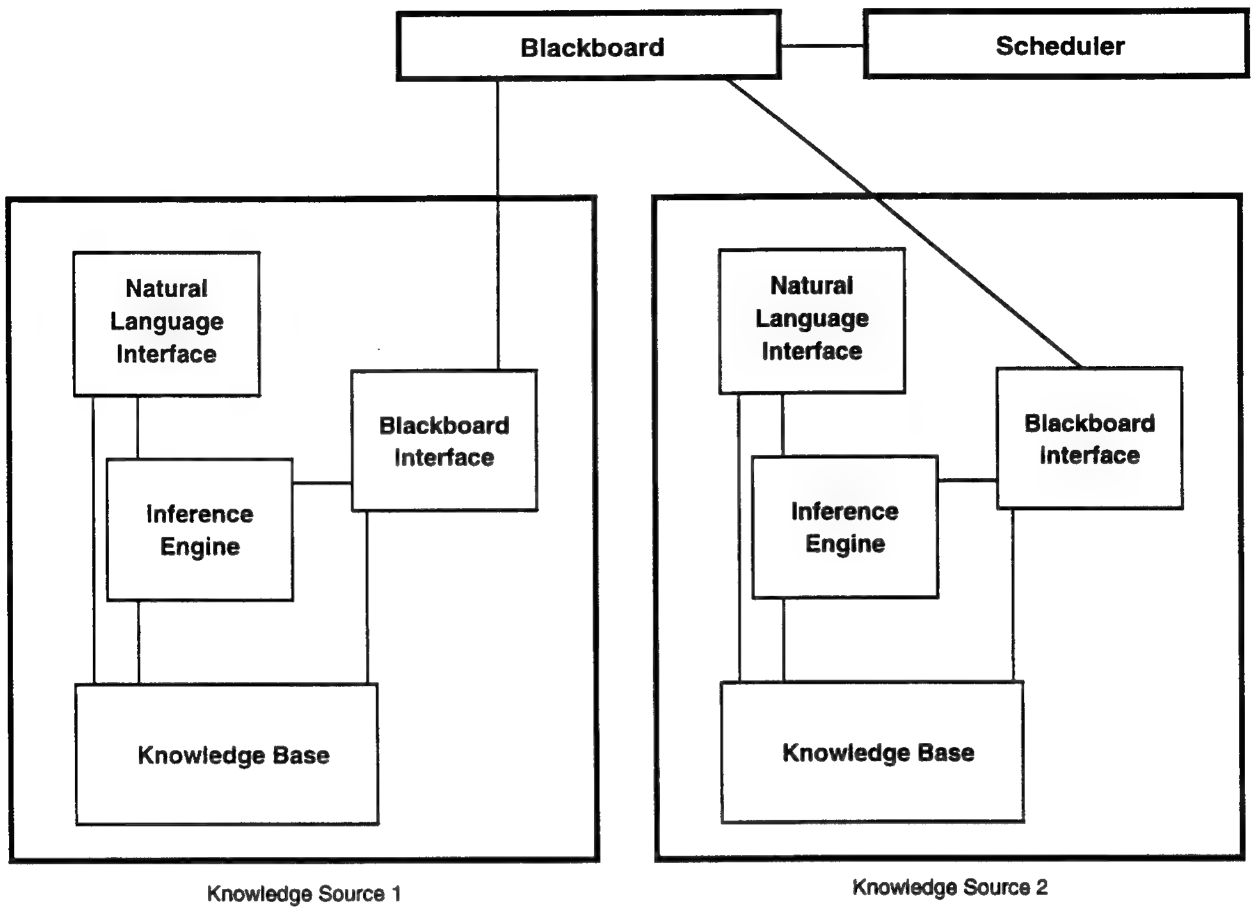


FIGURE 2.2-7. BLACKBOARD SYSTEM ARCHITECTURE

2.2.16.2 Blackboard System Control.

Blackboard systems determine their own cognitive behavior to solve problems involving the control of another system. The system must determine the problems it will attempt to solve and the knowledge and methods it will use to solve the problems. The system goal can be choosing an area of the blackboard on which to concentrate next, choosing a particular knowledge source to utilize next, or some action between the two. Whichever method is chosen, it must be the most efficient for the particular application. Selecting irrelevant knowledge sources must be minimized. The control can be integrated into the system design or it can be a separate ES.

The blackboard is a record of the hypotheses reached and the way they interconnect. Attached to each hypothesis on the blackboard is a measure of likelihood, which is a measure of the system's confidence level in the hypothesis. The likelihoods of the hypotheses are updated continually. Changes made to the blackboard are called events. These include creating or deleting objects, changing values, and creating or destroying links between objects.

Blackboard architecture is a control structure design. The knowledge sources can communicate only through the blackboard, so the events are tracked easily. Blackboard models contain a great amount of information. Knowledge sources are not activated in any particular order. They contribute hypotheses in response to appropriate situations on the blackboard, or the scheduler may decide when they should contribute in response to the situation. They represent an opportunistic approach to solving problems that allows the knowledge bases to contribute only when appropriate (Cohen 1985).

Advantages of blackboard programming techniques over other frameworks include the following:

- Dynamic control – Depending on the current state of the problem solving process, solutions may be formed incrementally.
- Focus of attention – Part of the emerging solution can be considered the next step.
- Flexibility – Knowledge can be used in many ways to form the solution; there is no preprogrammed method to obtain a solution.
- Islands – Pieces of the solution that are developed with a high degree of certainty need only to be connected to other pieces to build an overall solution to the problem.

Blackboard systems do not search for solutions; rather, they build an emerging solution from all independent knowledge sources. The presence of many different, independent knowledge sources leads to parallel processing architecture. Knowledge sources indicate to the blackboard when the problem solving process is terminated, either because an acceptable solution was found or the knowledge sources are depleted.

A blackboard framework is preferred over other methods of computer problem solving for several reasons:

- Modularity – Makes designing, testing, and maintenance easier.
- Dynamic control – Provides many capabilities for controlling the problem solving method.
- Efficiency – Provides maximum efficiency because the system selects the most appropriate knowledge to apply and focuses attention on the most profitable areas of the problem.
- Concurrency – Allows parallel processing because of the structure of the blackboard system.

2.2.16.3 Applications.

Following are problem types that are well suited to being solved by a blackboard system:

- Problems requiring many different and specialized types of knowledge.
- Problems with a natural hierarchy of the problem solving strategy.
- Problems that will benefit from solutions developing incrementally.
- Problems where limited data are available or for which only a partial solution is possible.

2.2.16.4 Explaining Actions.

An important feature of knowledge-based systems is the ability to explain their actions. The actions that are taken by knowledge bases are triggered by data or previously generated solution elements. The explanation of a conclusion should include the following:

- Evidence of the data on which the argument is based.
- A warrant, which supplies the connection between the evidence and the conclusion.
- A qualifier, which indicates the degree of confidence in the evidence leading to the conclusion.
- Evidence that supplies support for the warrant.

2.2.16.5 Future of Blackboard Systems.

More research is required to determine the types of applications that are suitable for blackboard systems. Also, further study of the characteristics of particular types of design choices for the systems is needed.

2.3 FUZZY LOGIC.

Traditionally, vague or imprecise concepts, such as tall, big, and warm, could not be represented on computers. Expressions such as these are used in our language constantly and the ability to represent these types of concepts is helpful when dealing with set membership.

The strength of fuzzy systems, and their reason for being developed, was their ability to manage imprecision. Fuzzy logic provides the opportunity to model conditions that inherently are imprecisely defined and deal with statements that are obscure or subject to different interpretations. It introduces a method of representing the degree to which an object is a member of a particular set (Miyamoto 1990).

There are many complex industrial processes that cannot be controlled satisfactorily by conventional computing methods. The use of rule-based control established on approximate reasoning provides an attractive alternative. Fuzzy logic was developed for complex control systems in which mathematical models were difficult or impossible to create. Fuzzy logic is an alternative to traditional set memberships. It was developed to express inexact concepts such as "normal height and weight." These concepts are used frequently in everyday language. However, they become a problem when attempts are made to express them using binary distinctions (Cohen 1985).

Fuzzy models provide a more flexible way to generate accurate solutions to problems than the rigidness of crisp set solutions. Data are represented symbolically, but they are processed numerically. Fuzzy logic is a precise and accurate method of reaching valid conclusions in control systems. Correctly implemented, it will sacrifice only unnecessary precision. Greater precision can be achieved by increasing the number of input and output values and increasing the number of rules (Brubaker[2] 1992).

2.3.1 The Basic Theory.

There are few situations that are strictly true or false. Fuzzy logic proposes that membership functions operate over a range between 0 and 1, where 0.0 is total non-membership and 1.0 is total membership.

The membership functions usually are graphed by using overlapping triangles; however, other geometric shapes are possible. Membership functions identify regions that will control the system. The purpose of overlapping shapes is to allow for partial set membership. The transition between being fully a member of the set and not a member at all is gradual. It is possible for an object to be a partial member of more than one set at the same time. Figure 2.3-1 illustrates a set of membership functions related to throttle movement. If the rotational speed of the throttle is -20 radians/second, it is a member of the set "Small Negative." The throttle's degree of membership within that set is the point where a vertical line intersects with the side of the membership triangle. That would be approximately .6 for a speed of -20 radians/second.

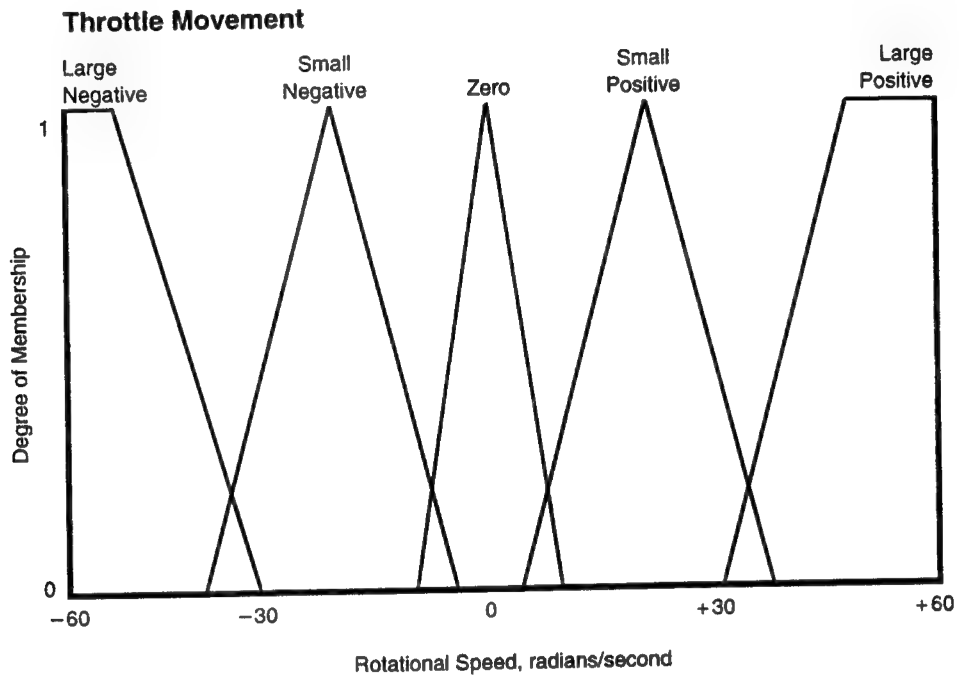


FIGURE 2.3-1. FUZZY MEMBERSHIP SETS

Fuzzy systems are based on the concept of parallel processing. Parallel processing allows large, complicated problems to be solved by breaking them down into smaller ones. The results of all rule processes are unified and expressed as a single logical sum. Compared to conventional control systems, fuzzy systems require fewer rules and reduce development time dramatically.

Rules are in the form of IF THEN statements that are linked by ORs. They use labels to describe the problem in words and are worded in a way similar to the human thought process. For example, a rule for controlling a flying aircraft may read "If nose is low and speed is high, then pull back on yoke."

A fuzzy system attempts to find a region that represents the intersection, union, or complement of the fuzzy control variables (Cox 1992). In the previous example, the control variables are nose and speed. They are the parameters that are being evaluated to control the aircraft. When they are both within a certain range (identified as "low" for nose and "high" for speed) at the same time, some resulting control action will be taken.

A frame of reference must be established for fuzzy systems to produce reasonable responses. For example, later in one context may be a very different time frame than later in another context.

One of the main advantages of using fuzzy logic is that modifications are made easily without the need to modify code. Rules and membership functions can be added without radical revisions. Therefore, designers can concentrate on the elements of the problem, as opposed to the details of a language such as FORTRAN or C (Williams 1992).

Fuzzy controllers are more flexible than conventional controllers. If some element, such as information, is missing, a fuzzy system will adjust itself accordingly.

2.3.2 Crisp Logic.

Traditional or crisp logic, such as Boolean, is based on predefined thresholds. For instance, in Boolean logic the output and input are always on or off; there are no other states. There are many situations that do not conform to Boolean logic. Fuzzy logic was developed for these situations (Lea 1989). The difference between crisp and fuzzy logic can be seen in figure 2.3-2.

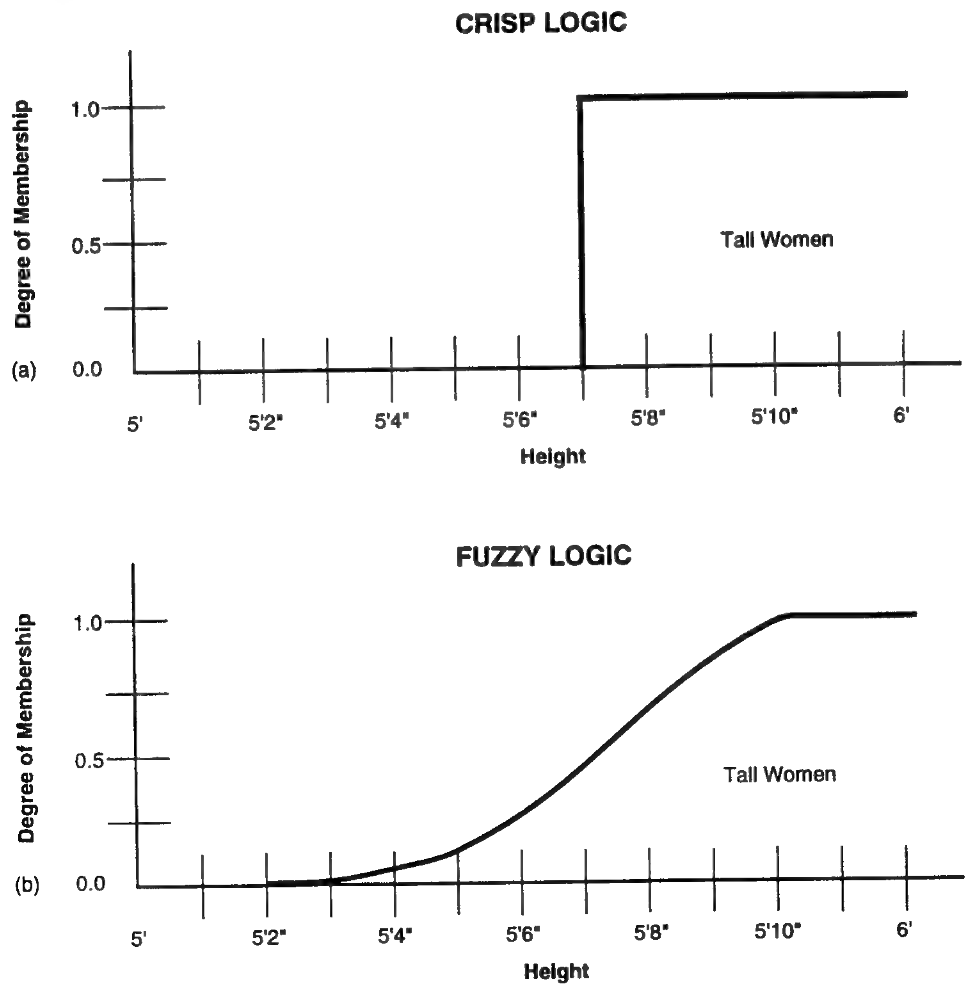


FIGURE 2.3-2. CRISP VERSUS FUZZY LOGIC

Thresholds in fuzzy logic are not clearly defined. Boundaries overlap each other and it is possible for an object to be partially a member of more than one set.

2.3.3 Fuzzy Control.

There are three steps followed by a fuzzy controller. They are fuzzification, inference, and defuzzification.

2.3.3.1 Fuzzification.

A fuzzy system takes inputs and fuzzifies them. Numeric inputs are converted to fuzzy values, such as small, fast, or warm. This is done using degree of membership functions, which are methods of determining the extent to which an object is a member of a particular group. For example, a particular aircraft may be considered both a cargo and a passenger aircraft, which would qualify it as a partial member of both sets. It could be more a member of one than the other, but nonetheless a member of both to some degree. The resulting degree of membership then becomes an input to the inference mechanism.

2.3.3.2 Inference Mechanism.

Inference is the process of inferring fuzzy actions by applying fuzzy inputs to a rule base (Legg 1992). Fuzzy actions include "pull back on the yoke," effecting a climb rate of 100 feet per minute. Rule-based systems are the most popular type of fuzzy system at this time. The inference process proceeds from the conditions to the conclusion (logical product) and then to the logical sum (Omron 1991). The logical sum is a combination of the results of all the rules. Figure 2.3-3 illustrates how a fuzzy system determines the control actions to be taken for a particular example. The object is to balance a stick in the upright position using seven rules:

- If the stick is inclined moderately to the left and nearly still, move to the left quickly.
- If the stick is inclined slightly to the left and falling slowly, move to the left somewhat quickly.
- If the stick is inclined slightly to the left and rising slowly, do not move much.
- If the stick is inclined moderately to the right and nearly still, move to the right quickly.
- If the stick is inclined slightly to the right and falling slowly, move to the right somewhat quickly.
- If the stick is inclined slightly to the right and rising slowly, do not move much.
- If the stick is slightly inclined and nearly still, do not move much.

There are seven possible sets in each of the two position antecedent blocks. In the left/right position blocks, negative corresponds to leaning towards the right and positive corresponds to leaning towards the left. There are 3 degrees of membership in either direction: small, medium, and large. In the falling / rising position blocks, negative corresponds to falling and positive to rising. There are also 3

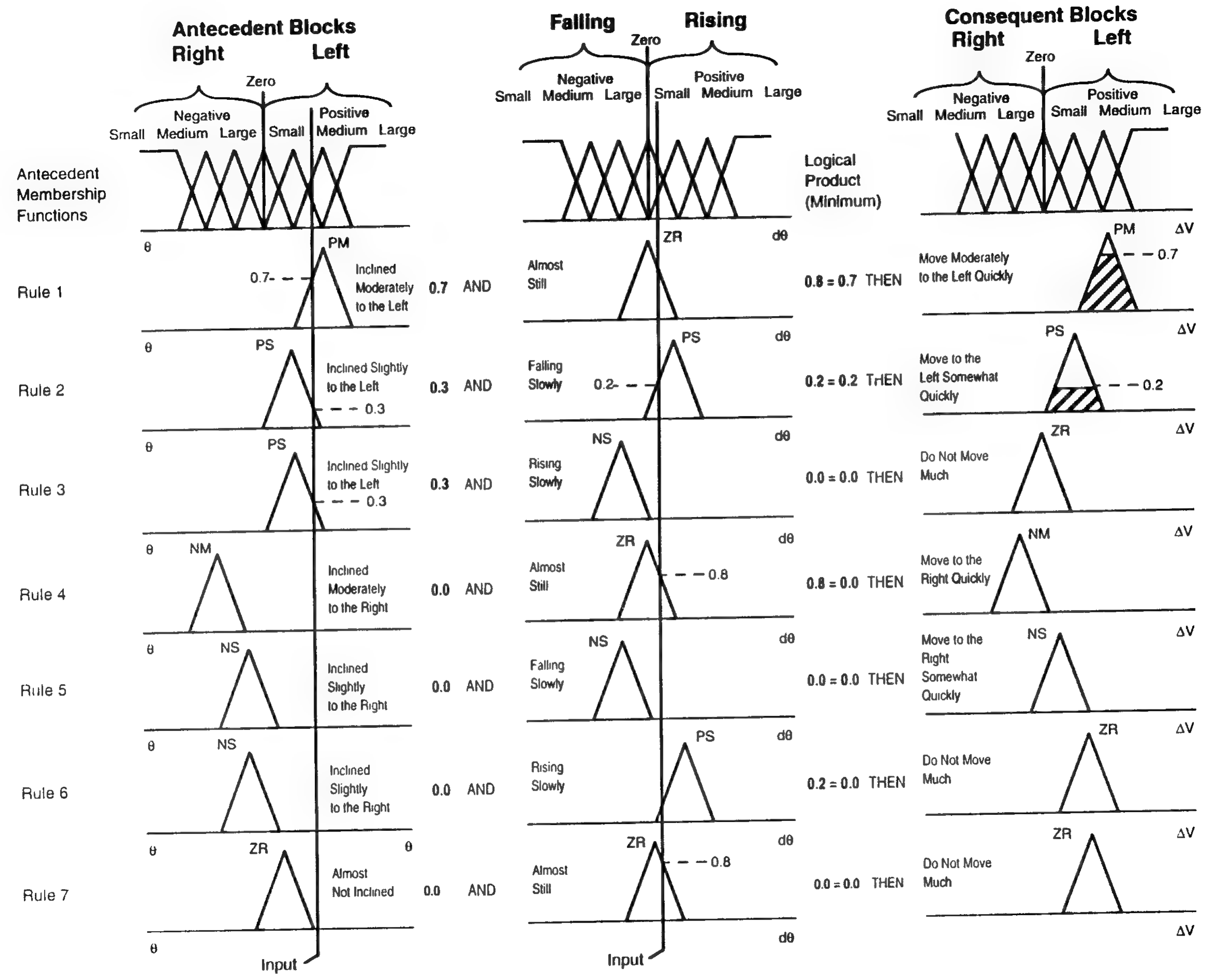


FIGURE 2.3-3. OBTAINING THE LOGICAL PRODUCT

degrees of membership in either direction: small, medium, and large. Both position blocks have a zero where the stick is balanced.

The line labeled "input" is one instance when the stick is at a certain point and the system is attempting to determine appropriate action. The point is a member of several sets. Therefore, it must be determined which rules to apply to the input. This can be resolved by evaluating of which sets the input is a member. As far as left/right position, at this point the stick is mostly a member of the set "inclined moderately to the left." It is also partially a member of the set "inclined slightly to the left." As far as falling/rising position, at this point the stick is mostly a member of the set "almost still" and partially a member of the set "falling slowly."

By performing an "AND" operation on the membership functions, the consequent blocks for each individual rule can be determined. These values are known as the logical product. Figure 2.3-4 shows the logical sum, which is the total result of summing the consequent blocks. This total is defuzzified by determining its center of gravity, which, in turn, becomes the output of the system. The output is generally applied to another system to initiate an action.

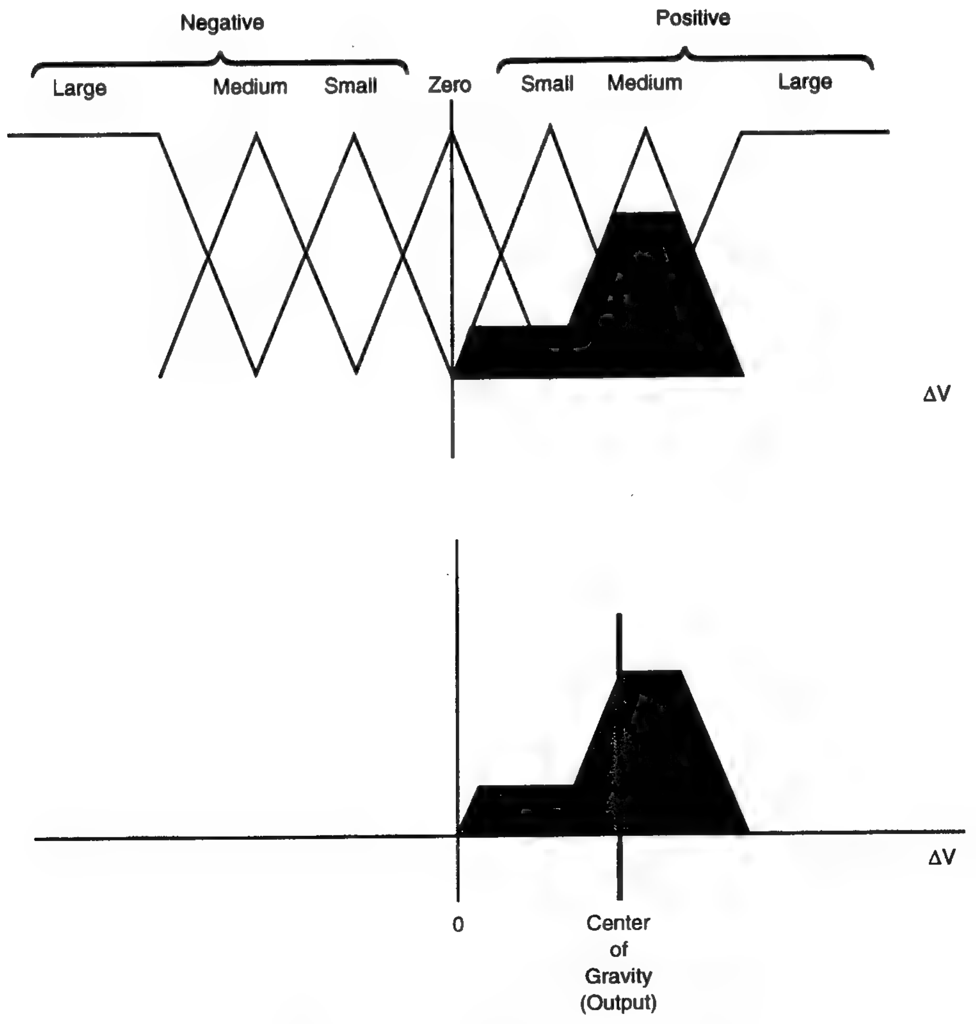


FIGURE 2.3-4. THE LOGICAL SUM
(as applied to the system in figure 2.3-3)

The inference process determines the degree of truth of a rule. The corresponding action correlates to this degree of truth. The inference system is a collection of fuzzy IF THEN rules. The inference mechanism uses a method of approximate reasoning by which an imprecise conclusion is deduced from a collection of imprecise premises (Lee 1991).

2.3.3.3 Defuzzification.

Defuzzification is the process of translating fuzzy or non-crisp values into values that are crisp and clearly defined. A defuzzifier converts inference process values to a fixed value. Generally, a non-fuzzy control action, such as a fixed voltage applied to another system to cause an action, is needed as a result of the inference process. Normally, more than one rule applies to any given set of inputs, so the fuzzy system must combine the results of several rules.

Defuzzification often is used in real world engineering events such as communications. When a signal is transmitted, it is affected by many outside influences, such as noise and electrical interference. At the receiving end, the signal must be defuzzified or cleaned up to determine what was actually transmitted.

A defuzzifier calculates the crisp output of the fuzzy inference by one of a number of methods (Brubaker[2] 1992):

- The centroid method calculates the center of gravity of the final fuzzy space which is formed by the sum of all of the rules in which the antecedent has been satisfied. This is currently the most popular technique.

- The singleton method is a special case of the centroid method. It uses weighted averages to represent each fuzzy output set as a single output value. This method requires less computation than the centroid method and can be modeled as a feedback controller by sampling selected outputs.

- The composite maximum method uses the result of the one rule that has the highest probability of being correct or the maximum degree of membership value from the various rules.

- The composite moment method assigns weights to all rules, based on the degree of membership values, and then averages these weights.

The method of defuzzification chosen by a designer depends on the time, memory, and degree of accuracy that are available. The composite maximum and composite moment are methods which require more testing to determine their accuracy and appropriate applications.

2.3.4 Control Systems.

Conventional control systems do not have the ability to self-adapt to changes in inputs. Adaptive fuzzy control systems can learn, explain their reasoning, and perform self-modifications, when necessary, to produce more accurate results. They can change supporting system controls, modify the characteristics of the rules, modify the topology of the fuzzy sets, and change the method of defuzzification. Their construction is similar to NN systems.

Fuzzy logic systems make excellent control systems. Controlling with IF THEN constructs is simpler than creating complex mathematical models. It is generally easier to create and understand fuzzy control rules than conventional control rules.

Fuzzy logic systems are designed to handle complex requirements using controllers that are simple, inexpensive, and easy to maintain. Because fuzzy systems use intuitive terms, complex systems are modeled easily and rapidly. A system's operational and control laws are expressed linguistically. Figure 2.3-5 shows the major components of a fuzzy logic controller.

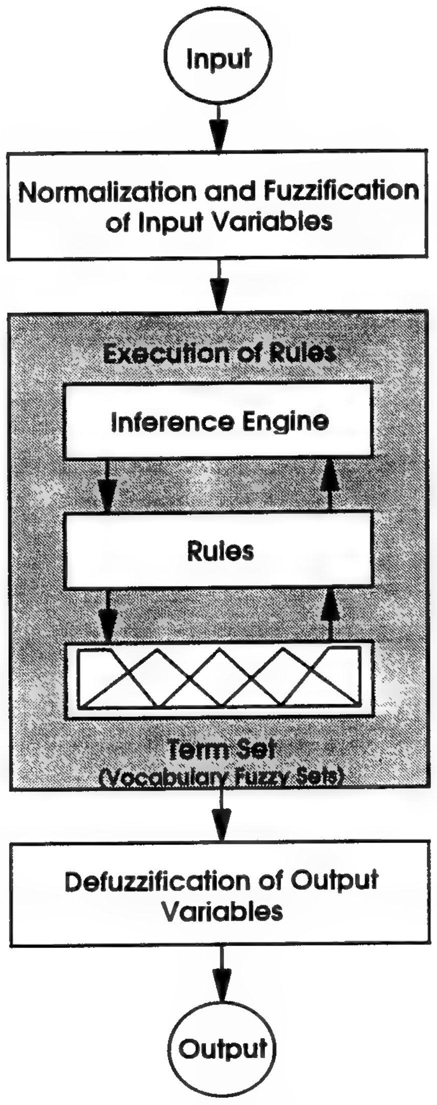

Input
Normalization and Fuzzification of Input Variables
Execution of Rules
Inference Engine
Rules
Term Set (Vocabulary Fuzzy Sets)
Defuzzification of Output Variables
Output

FIGURE 2.3-5. FUZZY CONTROLLER

Fuzzy systems that are not integrated with NNs can adjust their behavior according to the previous set of outputs. They cannot reorganize themselves or change the rules that they are using. Reorganization can be done with the assistance of an NN integrated into the system.

Fuzzy associative memory systems associate the input of a fuzzy controller with the desired output, similar to an NN operating in unsupervised mode (Cox 1992). Fuzzy systems, like NNs, can run in supervised or unsupervised mode. They are more likely to run in unsupervised mode because they rely heavily on clustering techniques. Clustering techniques, which are sometimes used by NNs, are methods by which systems establish groups of related data.

Fuzzy control systems can be cheaper, easier to develop, and more accurate than traditional control systems. Also, they can respond faster and consume less power than traditional digital methods. These factors provide the opportunity to bring products to market faster and more cost effectively.

2.3.5 Probability.

Many people confuse fuzzy theory with probability. Both operate over the same numeric range, where 0.0 represents false and 1.0 represents true. A statement is probabilistic if it contains some likelihood or degree of certainty, i.e., the likelihood that an event will happen. Probability theory deals with the uncertainty that results from random behavior. The difference between probability and fuzzy theory is that probability states the chances of membership, while fuzzy logic states the degree of membership or the degree to which an event has happened. Fuzzy theory deals with the uncertainty associated with boundary conditions (Brubaker[1] 1992).

For example:

Probability: There is an 80 percent chance that after pre-flight checks, the aircraft is safe to fly. This statement indicates the chance of the aircraft being safe to fly.

Fuzzy Logic: The aircraft's degree of membership within the set of safe aircraft is 80 percent, and the aircraft's degree of membership within the set of unsafe aircraft is 20 percent. This statement assumes that the aircraft is more or less safe, and indicates the degree of safety.

2.3.6 Classification.

Classifying data that are input to a fuzzy system is an integral component of the system's operation.

Fuzzy systems use the clustering techniques of NNs to determine fuzzy membership functions (Klimasauskas 1992). The fuzzy membership functions are the degree of membership functions used to determine set membership. When using fuzzy classifier design, the output is crisp; however, fuzzy techniques are used during processing to achieve the desired output.

2.3.7 Parallel Processing.

Fuzzy systems can use parallel processing, which allows multiple rules to be processed simultaneously. High-speed processing can be attained with hardware controllers because the rules are processed independently of each other. Errors will not drastically affect the final result in a parallel processing architecture.

2.3.8 Hardware.

Fuzzy systems do not require massive computing power and many applications do not require special fuzzy Integrated Circuits (ICs). There are three hardware options in a fuzzy system (Legg 1992):

- Existing microcontrollers
- Enhanced existing microcontrollers
- Dedicated fuzzy processors

Normally, fuzzy processors work along with a general purpose microprocessor, which is needed for functions such as preprocessing inputs, keyboard interaction, and support in other input and output areas. Tools that can assist in designing fuzzy systems range in price from a few hundred to a few thousand dollars (Legg 1992).

Simple fuzzy controllers require less than 10 rules, while complicated ones may require 40 or more. Some fuzzy processors allow arbitrary shapes for the membership functions. Some limit the number of shapes and others allow triangles only.

Fuzzy logic systems can be implemented with analog hardware; however, the fast and inexpensive processors available today handle analog signals better when using digital techniques (Brubaker 1993).

2.3.9 Designing a Fuzzy System.

Fuzzy logic does not require that a designer be able to model a control problem mathematically. All that is required is a firm understanding of the problem, a basic understanding of physics, and a concept of how the system should behave. The designer is not restricted to what can be described with equations (Conner 1993).

The designer of a fuzzy system needs to identify a number of elements:

- The input, output, and operations to be performed on the data.
- The control and solution variables, fuzzy regions, and labels associated with those variables.
- The rules connecting inputs to outputs.
- The method of defuzzification.

Following are rules of thumb in fuzzy system design (The Huntington Group 1993):

- Understand the system, its requirements, and the available tools.

- Ensure system stability by verifying that the plot of system outputs, as a function of the system inputs (response curve), remain smooth around the system's operating point. A fundamental characteristic of fuzzy logic is the ability to handle nonlinearities; however, sharp spikes in the response curve should be avoided.

- Carefully identify the variables that affect the system and the ranges of values into which they may fall. These variables serve as inputs to the rules.

- Define rules by identifying controller actions at clear, well-defined operating points. Account for all possible input conditions.

- Design a fuzzy system incrementally.

- Be flexible with the rule base. For example, a system may have more than one rule base.

- Treat time as a fuzzy variable in real-time systems. For example, system response time may be small, medium, or long. System response, as determined by the rule base, will be different based on the value of response time.

- Perform many simulations.

Control variables appear in the premise of a rule (e.g., density, speed) and are linked by AND. Determining these variables and creating the rule base are the most difficult part of designing a fuzzy system.

Labels are names that are attached to fuzzy sets or membership functions (e.g., cold, warm, hot). The shapes and slopes of membership functions can be changed at any time to meet a design criterion. Overlapping input functions guarantee that at least two rules will fire for any given input and ensure a smooth output curve. Generally there are an odd number (between five and nine) of labels (e.g., slow, medium, fast) associated with each variable (Cox 1992). Membership function sets should be grouped closely at points where the most activity in the system being monitored occurs.

"Hedges" are terms such as very, more or less, somewhat, and sort of. These terms modify fuzzy values. Normally, they precede a label to emphasize it. Fuzzy systems have the ability to define hedges. This adds to their flexibility.

A membership function is interpreted by a fuzzy controller as a family of crisp sets, called an alpha-cut, and a fuzzy set is interpreted as a family of alpha-cuts. The alpha-cut is a fundamental operation that relates the fuzzy sets to a family of crisp sets. The alpha-cut is needed to transform fuzzy sets into crisp sets (Miyamoto 1990).

Rules in a fuzzy system normally are determined by an expert. They are stored in a rule base that is similar to the thought process that an expert would use to solve a problem. Because fuzzy rules are abstract, they do not need to cover every combination of cases that is possible. The rules are defined by the relation between antecedent and conclusion.

2.3.10 Fuzzy Logic Advantages.

There are a number of misconceptions associated with fuzzy logic. Most of these are addressed in the following list of advantages (Cox 1992):

- Fuzzy logic allows partial or gradual degrees of membership. The concept that fuzzy logic provides imprecise answers is a common misconception.

- Fuzzy logic is different from probability. When dealing with probability, the event is clearly defined. Fuzzy logic is concerned with the ambiguity in the description of the event.

- Fuzzy sets are easy to conceptualize and they are modeled in the same fashion as an expert's thought process.

- Fuzzy systems are stable, easily tuned, and can be conventionally validated. There are fewer rules in fuzzy systems than in conventional control systems so tuning is simpler.

- Fuzzy systems are different from, but complementary to, NNs. Fuzzy systems are similar to NN classifiers. However, in a fuzzy system the classification processes performed on the data are more flexible for the developer.

- Fuzzy logic is more than process control. Fuzzy logic provides a way of representing and analyzing information, independent of particular applications.

- Fuzzy logic is a representation and reasoning process, not the answer to all AI problems. Fuzzy logic provides the opportunity to manage concepts that are outside of the realm of conventional Boolean logic, but it is not the final solution to all problems.

2.3.11 Fuzzy Logic Disadvantages.

Although fuzzy logic is a promising alternative to traditional control systems, there are a number of factors that need to be addressed:

- There is not a clear understanding of what applications are appropriate for fuzzy systems and how they compare to traditional designs.

- Fuzzy systems, when optimized for speed, use an amount of microprocessor memory comparable to traditional control methods.

- There is no formal design method or system modification and tuning method that will eliminate all unnecessary rules to find the optimum rule set. The systems are not analytic and system stability cannot be proven.

- Fuzzy systems can be designed more quickly than conventional control systems. However, they require more simulation and tuning to optimize their performance. Fuzzy systems may require the same amount of development time as traditional systems.

- Fuzzy systems may not be the best alternative for all control system applications because they cannot prove closed-loop system stability. This requires much simulation and testing. If traditional control methods are adequate for a particular system, there is no need to change methods. In general, if a system can be defined using conventional control equations, fuzzy logic is not necessary.

2.3.12 Applications.

Applications that have used fuzzy logic systems include decision support systems, financial planners, diagnostic systems, and information retrieval systems.

Types of systems that can benefit from fuzzy logic control include the following (Omron 1991):

- Non-linear systems
- Systems with insufficient or unclear data input
- Systems that are difficult to control
- Systems that benefit from or are normally controlled by human intuition
- Systems that require adaptive signal processing because of dynamic environmental conditions
- Systems with multiple inputs or conflicting constraints
- Systems for which models are unclear or difficult to define

Fuzzy logic controllers have been used successfully in a wide arena of applications. Items such as car cruise controls, washing machines, elevator controllers, vacuum cleaners, subway engine controllers, and security investment systems have benefitted from fuzzy systems (Schwartz and Klir 1992).

Fuzzy logic systems are well-suited to model very large systems, natural or man-made, such as weather, oceans, economy, and stock markets (Brubaker 1993).

2.3.13 Future of Fuzzy Logic.

Designing with fuzzy logic is regarded with skepticism, in part due to its close association with AI. Another reason for skepticism may be distrust of a new technology that is perceived as revolutionary. Even its unusual name has added to the skepticism.

However, when compared to traditional system development, fuzzy systems can be designed more easily than systems using a linear control approach. Also, the basis of fuzzy design is easier to learn than linear methods.

Japan and Europe already utilize fuzzy technology in practical systems. In the United States, companies are beginning to turn to fuzzy logic as a viable option to solving many types of problems. Competition will encourage more involvement in this new technology.

Replacing traditional control methods with fuzzy systems may expand a system's capabilities. Just as the microprocessor revolutionized the electronics industry, fuzzy logic has the potential to do the same for the control industry.

2.4 NEURAL NETWORKS.

NNs are information processing systems that can be trained to solve problems. Artificial NNs are constructed to resemble the biological neurons of the brain. Hundreds of thousands of simulated neurons are linked together by synaptic connections. The structure of an NN is shaped and reshaped, depending on associations among facts.

NNs have been described as the second best way of solving a wide range of problems. They are capable of solving virtually any problem that involves mapping input data to output data. They produce relationships, rather than insight. The network maps the array of inputs to the array of outputs. NNs can find acceptable answers, although not necessarily the best answers, in a reasonable amount of time and for reasonable cost. NNs can spare tedious and expensive software design when compared to other methods of solving problems. NNs use information in large databases to extract a set of rules (Intel[2] 1990).

Rapid progress is being made in the field of NN technology. Scientists are trying to duplicate the way the brain makes inferences from incomplete or confusing information (Tazelaar 1989). NNs can classify, store, recall, and associate information. They can provide several recommendations for a particular situation in a few seconds. A machine that emulates the way a brain operates must work in a parallel fashion. There is no centralized site of computing power in the brain. NNs do not have a centralized computing site either. Rather, each individual computing site works independently.

Several factors make NNs an attractive system choice for appropriate applications. These include using incomplete data to produce approximate results; managing large amounts of data quickly and efficiently; and using parallelism, speed, and trainability for fault tolerance (Obermeier 1989). However, appropriate applications for NNs still are being researched and clarified. The main application at this point in development is pattern recognition. NNs are best suited for problems in which overall accuracy may be less important than estimation and flexibility.

NNs learn by association or examples, rather than by rules or mathematical formulas. Neurocomputers are not programmed, they are trained. One of the most important objectives when developing an NN is to determine the problem and the types of data that are available. Once this is accomplished, the next step is converting the information to a form that the NN can understand.

Two significant elements in NNs are learning and recall. Learning is the adjustment of weights (the strength of the relationship between elements in the NN that are computing outputs) to store information. Recall is the ability to retrieve the information stored in the weights.

2.4.1 Learning.

There are a number of factors that must be considered when training NNs:

- Presentation of exemplars (a collection of correlating input and output data sets)
- Update interval – Update after each individual or after the whole group
- Type of supervision – Supervised or unsupervised learning
- Type of updating – Real-time or non real-time updating
- Number of tasks to be learned

NNs are data driven. While specific algorithms and problem definitions do not have to be formulated, the data must be accurate. If the data contain errors, the network will be trained according to those errors and will not be capable of producing accurate results. Although the success of an NN depends on accurate data, a clear physical understanding of the problem is not required to train the NN.

Following are characteristics inherent in NNs:

- Theory or software bottlenecks are eliminated.
- The model can be as complex as required.
- Emphasis is on quality of data.
- Sufficient numbers of examples are required for training.

Training is one of the most appealing qualities of NNs. Training is the process of presenting inputs and correlating outputs to the NN so that it will adjust connection weights according to the Input/Output (I/O) relationship of the data. NNs are shaped and reshaped, depending on the information that is received and internal states of the system (Rocha, et al. 1992). Learning is accomplished by firing neurons, which are the computing sites in an NN, and adjusting the connecting weights between neurons. A neuron fires when the value of the incoming signals reaches a certain threshold within that particular neuron. When a neuron fires, it sends an outgoing signal to surrounding neurons.

The most popular learning method is back propagation. To solve a problem with a back propagation network, the system requires many sample inputs with desired outputs. The network learns by adjusting weights. Back propagation compares actual prediction with a correct prediction and uses the error to change adaptive weights in a direction that is error reducing (Carpenter and Grossberg 1992). Known information is presented at the input, weighted values are assigned to the connections, and then the network is run repeatedly until the output is accurate. Running the network will readjust the connection weights. Data associations that are frequent will strengthen useful neural couplings, weaken negative couplings or connection weights, and disconnect unrelated ones. The network must be trained on preselected inputs and outputs and then it can be run on new information.

It is possible to overtrain an NN. Larger networks contain a greater number of free parameters than smaller networks because they have more input and output variables. The number of nodes in the network is determined by the number of variables in the problem. If a network contains many variables, there is more chance of errors occurring in the training data sets. The increased numbers of variable parameters in an NN leaves more opportunities for errors to occur during the training process.

The manner in which the data are represented when it is presented to the network affects the network's ability to learn and generalize (Anderson 1992). Knowledge representation in an NN is obtained through the structure of the association of neurons, as well as through the properties of individual neurons. Information in an NN cannot be extracted or found at any particular address. Information is not simply stored in a neuron. To use NNs in the most efficient and successful manner, a full understanding of the technology and its applications must be achieved. On average, 81 percent of the time required to develop and deploy an application is spent acquiring, analyzing, and transforming data (Klimasauskas 1992).

2.4.1.1 Supervised Learning.

In supervised learning, trial and error inputs are used to teach the network correct and incorrect responses. Learning is accomplished through direct comparison of the network output with known correct answers. When an error occurs, an error signal is sent back through the network and connection weights are changed to prevent the same error from recurring. This type of learning requires *a priori* knowledge of the results. Supervised learning uses class membership information. It detects errors because of pattern misclassifications. A trainer supervises what is output in reference to the input, and the network establishes a general representation of the regularities of the data.

2.4.1.2 Unsupervised Learning.

Unsupervised learning involves no trainer. Rather, the network organizes itself by establishing its own classification of inputs. This is accomplished by exposure to a number of inputs. Information used by the system is not categorized. Therefore, unsupervised learning cannot detect errors as well as supervised learning. Although this method is less computationally complex, it usually is less accurate.

The learning goal of unsupervised learning is not defined in terms of specific correct examples. The only available information is in the correlations of the input data or signals. The network creates categories of data and procedures and produces output signals corresponding to the input category. The creation of the categories of data is referred to as clustering.

Clustering is also known as unsupervised learning because the system is learning the correct labels for subgroups. Clustering is a branch of pattern recognition. The input layer in clustering nets is connected directly to the output layer. Two primary uses of clustering techniques are determining fuzzy membership sets and developing classification systems for case-based reasoning.

2.4.1.3 Self-Supervised Learning.

Self-supervised learning takes place without an external monitor. There is a feedback device that detects errors and adjusts weights accordingly. This is a combination of supervised and unsupervised learning techniques.

2.4.2 Transfer Functions.

The transfer function defines the value of the output signal (Lawrence 1991). The transfer function is a nonlinear function performed on the output of an NN. Different types of NNs use different nonlinearities, depending on the type of output desired. They are methods of limiting the output. There are five transfer functions regularly employed by NNs: linear, step, ramp, sigmoid, and gaussian. The differences among them have to do with the shape of the function. An NN system designer will choose the shape most appropriate for the particular application.

2.4.3 Construction.

A biological neuron consists of axons (output), dendrites (input), and synapses (pathways for transmission of impulses from one neuron to another). In an artificial NN, wires are used for axons and dendrites, and resistors with weighted values are used for the synapses (Obermeier 1989). NNs consist of layers of Processing Elements (PEs), weighted connections, and threshold functions. Figure 2.4-1 shows the NN architecture.

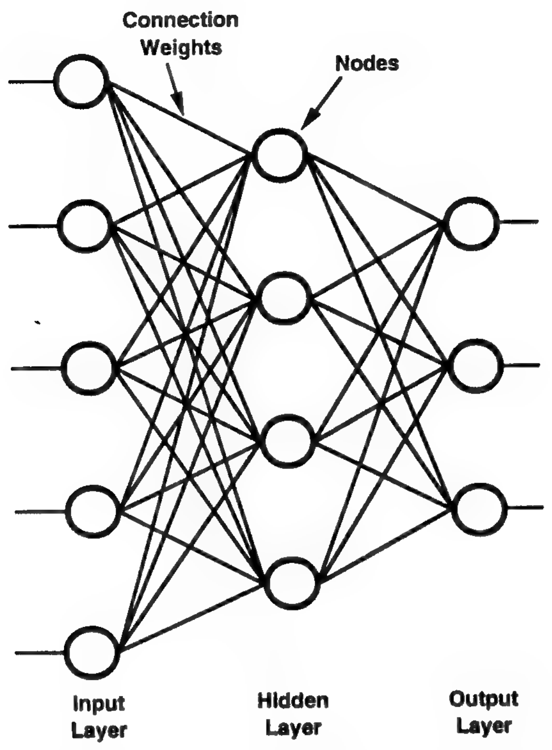


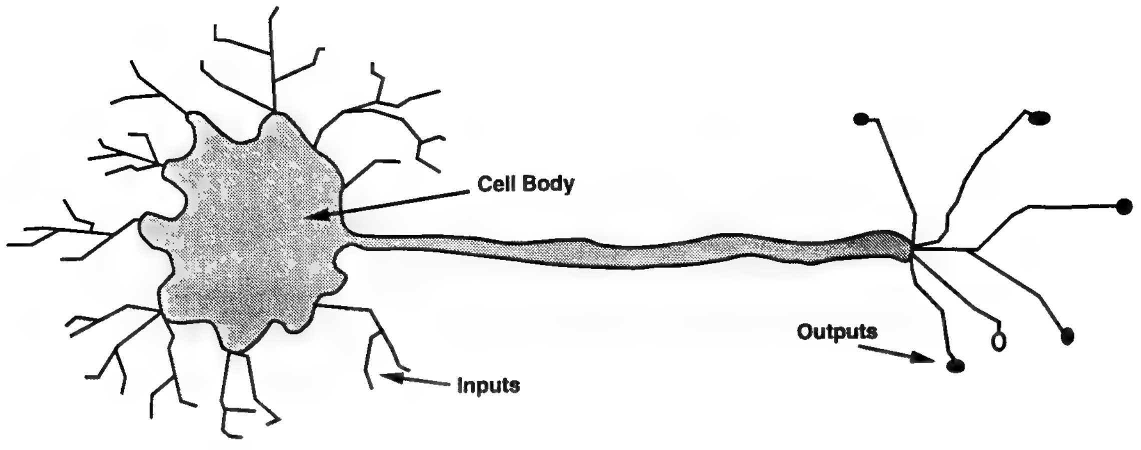
FIGURE 2.4-1. NEURAL NETWORK

The PE, or neuron, is the part of the NN where computing is performed. While NNs are inspired by biological neuroscience, they do not attempt to be biologically realistic in detail. The brain is a complex system and artificial NNs are comparatively simple. In designing an NN, humans are attempting to duplicate many of the desirable features found in the brain (Hertz, Krogh, and Palmer 1991):

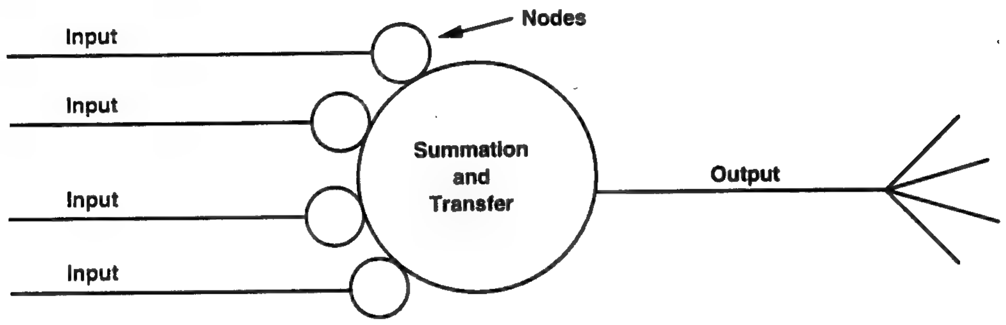
- Robustness and fault tolerance
- Flexibility
- The ability to manage fuzzy, noisy, or inconsistent data
- The ability to perform parallel processing
- Compactness, small size, and low power dissipation

Figure 2.4-2 shows the difference between artificial and biological neurons.

NNs operate efficiently in parallel because each PE operates independently of the others. All PEs operate in parallel, as opposed to traditional digital processors, which perform sequential arithmetic and/or symbolic information processing (Krogmann 1991). A large number of interconnected neurons work toward a common goal. A PE depends on adjoining inputs from abutting connections to produce its outputs. Traditional computers use a serial processing method to manipulate data. The CPU and the memory are two separate entities. When data needs to be processed, it must be called into the CPU, manipulated, and then returned to the memory.



Biological Neuron



Artificial Neuron

FIGURE 2.4-2. BIOLOGICAL AND ARTIFICIAL NEURONS

A network achieves stability, or converges, when the weight values associated with the PEs have finished changing. Networks normally converge, with training, to a stable solution.

Most NNs use the McCulloch-Pitts model of a neuron. The neuron fires when the weighted sum of the inputs reaches or exceeds the programmed threshold. Artificial NNs obtain high speeds by running in a parallel manner. Traditional digital computers have high processing speeds, however, they execute instructions sequentially.

2.4.3.1 Network Connections.

The way in which neurons are connected determines the type of processing that will occur. A connection is a line of communication that goes from a sending neuron to a receiving neuron. Connections may go between the layers of neurons in an interlayer or intralayer fashion. The connections modulate the information flow between the PEs. The strength of the connections between neurons is a weight. A weight controls the strength of the incoming signal to the neuron. The degree of cell stimulation is determined by the frequency of the signals coming into the cell (Krogmann 1991). The collection of weights for the entire network is called a weight matrix. The knowledge of the network is distributed across the interconnections of neurons. Weights can be positive, negative, or zero. Feedback occurs when the output of one layer goes to the input of the previous or same layer.

The network interconnections can be changed by generating new connections, losing existing interconnections, and changing existing interconnection weights.

Detailed past information is stored in the connection weights of the network and is represented by a matrix.

2.4.3.2 Neural Network Layers.

A layer is a collection of neurons in an NN that is separate from other collections. Neurons are located in input, output, or hidden layers. Hidden layer neurons connect only to other neurons. Input and output layer neurons connect to the environment outside of the NN. Three layers are sufficient for any arbitrary non-linear mapping.

The input layer does not process data; it passes the inputs to the hidden layer. The hidden and output layers are the processing layers.

The NN designer determines the number of nodes in a hidden layer primarily by trial and error. Nodes are added until they no longer produce any benefit for achieving a solution.

Determining the optimum number of nodes that the hidden layer should contain often requires experimentation. The hidden layer sorts the data and determines which inputs are associated with which outputs. The number of nodes in the input and output layers is dictated by the problem requirements.

Every neuron in one layer is connected to every neuron in the next layer (Lawrence 1991). The hidden layer makes connections between inputs and outputs. Each PE collects values from all of its input connections. The inputs to a neuron are the outputs from surrounding neurons. A neuron sends a signal when the sum of the input signals has reached a specific level. Eventually a pattern is formed by the internal neurons. The neuron then calculates the weighted sum of all the inputs and produces a single output value. The NN recalls information by performing update operations for each of the PEs.

In most cases, three layers of PEs are sufficient to perform an operation. Normally, this can be accomplished with less than 500 neurons and 30,000 connections. More complex NNs can solve more complicated problems. However, they have many more connections for which weights must be calculated. Single layer NNs are used for pattern completion, noise removal, optimization, and contrast enhancement, but their representations are limited (Rothman 1993). Multi-layer NNs are used for pattern classification, pattern matching, and function approximation.

2.4.4 Hopfield Neural Network.

One of the most popular NNs is the Hopfield model. It is an associative memory network, which means that it is a filter. The output is the same as the input without the noise. This type of network is used mainly for pattern recognition problems. A number of patterns are stored in the network. These act as attractors for the inputs. Each stored pattern has a basin of attraction. When patterns are input into the network, they will be attracted to the stored pattern in the appropriate basin of attraction. Figure 2.4-3 depicts the basins of attraction associated with the stored patterns in an NN.

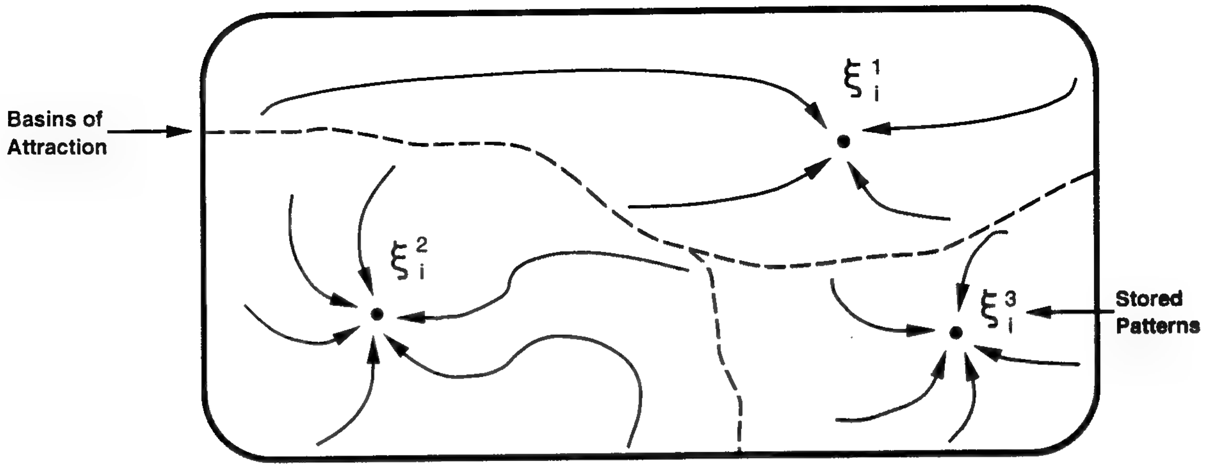


FIGURE 2.4-3. ATTRACTORS IN A NEURAL NETWORK (Hertz 1991)

This type of network is dynamic. In a dynamic system, energy continues to decrease with time until it reaches a stable point. Outputs are fed back as inputs until the weights no longer change. At that point, the network has converged. Figure 2.4-4 shows an NN with feedback connections. Associative type networks always converge. Every time a bit changes, it is headed toward a lower energy state. If a system is decreasing in energy, it is headed towards stability. If a system is increasing in energy, it may be headed towards instability.

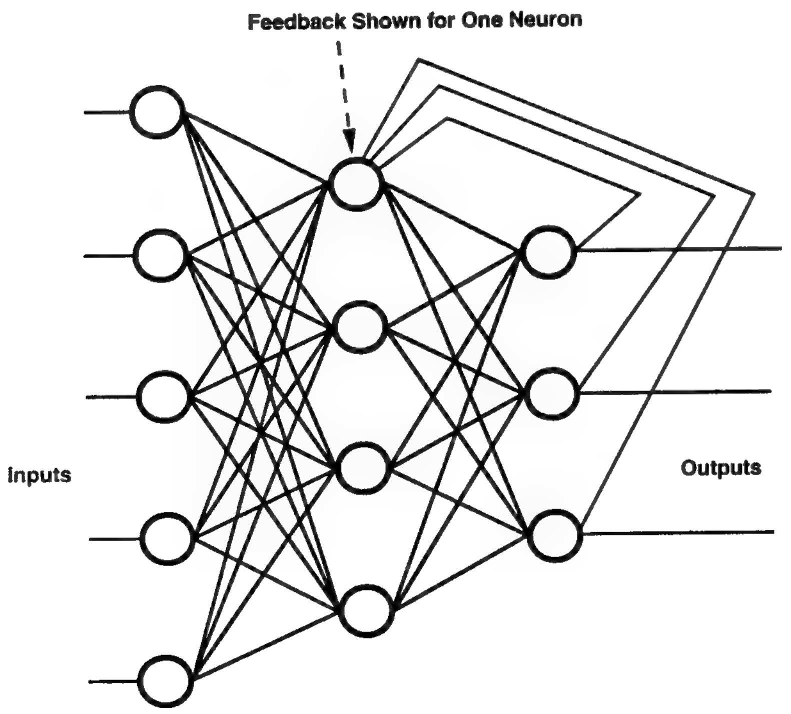


FIGURE 2.4-4. A FEEDBACK NEURAL NETWORK

It is possible for an input to converge to a spurious state. A spurious state is a pattern that is not one of the stored patterns, but acts as an attractor. Spurious states occur for various reasons. The most common reason being that the reverse of the stored states are minima and have the same energy as the original patterns. There are also states that correspond to a linear combination of an odd number of patterns. The spurious states have small basins of attraction relative to the stored patterns. If an input too closely resembles the small basin of attraction associated with a spurious state, then the input may converge to that state. There are various methods to reduce or remove spurious states, such as adding one known bit to each stored pattern and always checking that bit at the output. If the bit is incorrect, this implies that the network has converged to a reverse stored pattern and every bit in the pattern must be changed for the correct answer. There is a limit on the number of patterns that can be stored in the network while maintaining credible results.

The Hopfield NN can match patterns from distorted images or images that are embedded in other patterns. The most important NNs for pattern recognition are clustering nets and classifier nets.

There are several variations of the Hopfield network. The particular application will dictate the best choice. There are other, less popular types of networks, such as the heteroassociative, in which the inputs and the outputs are different. The objective is to make a prediction that is not easily measured.

2.4.5 Designing a System.

Following are some issues that must be resolved before an NN can be successfully implemented (Hertz, Krogh, and Palmer 1991):

- Architecture – How many units and layers are necessary? How should they be organized and what type of updating should be used?

- Training – How many examples should be used? How many times should the network cycle through the examples? Should learning be supervised or unsupervised, real-time or non real-time?

- Network type – How much can the network learn? How fast and robust is the network? How well can it generalize?

- Network hardware – What are the advantages and disadvantages of network hardware, and how does the hardware compare to software simulation.

Most work in neural computation has been done by simulating networks on serial computers. Very Large Scale Integration (VLSI) ICs traditionally have lagged behind the software models, but that is rapidly changing. Figure 2.4-5 shows one example of how an NN might be constructed. The Operational Amplifier (Op Amp) is a circuit element in which the output is determined by the difference in the input voltages. A Schmitt Trigger is a circuit element that exhibits hysteresis. Hysteresis is implemented with feedback, which holds the device at one of the two output states (on or off) unless a sufficiently large input is applied to overcome the feedback. The output of the trigger at any particular time does not depend only upon the present value of the input, but also on the past value. An important application of the Schmitt Trigger is the square-wave generator. A pulse type waveform can be generated from a continuous input (Savant, Roden, and Carpenter 1987).

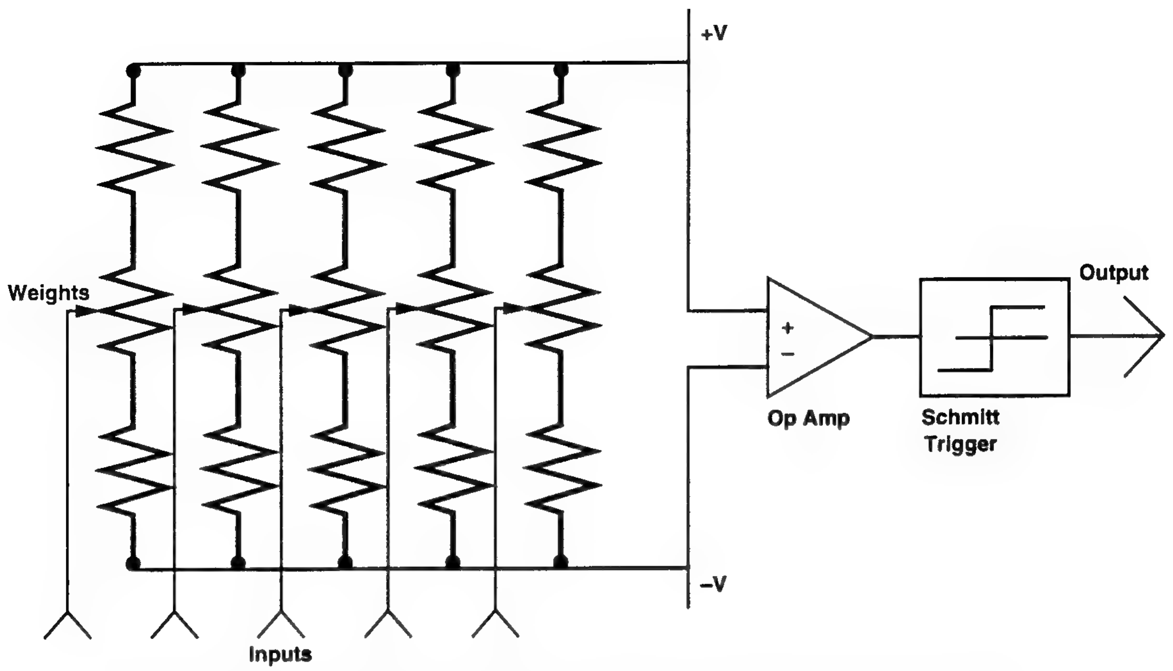


FIGURE 2.4-5. CONSTRUCTION OF A NEURAL NETWORK

Nodes must be updated to retrieve outputs. Network updating can occur synchronously or asynchronously. Synchronous updating is the process of updating all units simultaneously at each time step. At one given time, all outputs are computed and then returned as inputs according to a clock. The asynchronous updating scheme is similar to the manner in which the brain updates. In the brain, neurons update themselves whenever they are ready. Asynchronous updating in an NN can occur in two ways: at each time step, a random unit is selected to be updated, or each unit independently chooses to update itself with some constant probability per unit time (Hertz, Krogh, and Palmer 1991).

2.4.6 Applications.

NNs are used to solve problems that digital methods cannot handle efficiently. They can solve nonlinear problems because the neurons are highly interconnected and provide a close nonlinear relation between the input and output. Also, they can handle large amounts of data. Problems that involve complex mathematics and/or require quick, nonexact answers are suited for NNs. Also well-suited for NNs are problems that include many examples, but no rules or formulas. An advantage of NNs over rule-based programming is that if the process being analyzed changes, all that needs to be done is to collect new examples and retrain the NN. There is no need to determine new formulas and rewrite software.

The main operations that NNs perform include the following:

- Classification
- Pattern matching
- Pattern completion
- Noise removal
- Optimization (solution to a problem)
- Control

The main application is pattern recognition. NNs can recognize patterns even when the data are noisy, ambiguous, distorted, or variable.

Following are characteristics of appropriate problems for NNs:

- The algorithm to solve the problem is unknown or expensive to develop.
- Rules to solve the problem are unknown or difficult to define.
- Large amounts of data are available for the problem at hand.

NNs, however, are not suitable for handling all types of problems. They are not suited for number processing, therefore, they should not replace data-based or knowledge-based processing. Traditional digital computers are better suited for problems that require accuracy and computational power. NNs are not an extension of sequential computers. Problems that are linear should use linear problem solving techniques.

There are six basic steps required to prepare an NN (Lawrence 1991):

a. Define the problem.
b. Select network inputs and specify the output.
c. Decide how to represent the information and gather it.
d. Define the network.
e. Train the network.
f. Test the trained network.

Precision is the strength of conventional computers. Although NNs are not precise, this does not imply that they are inaccurate. They are accurate within the defined bounds of precision. NNs are not suited for deduction or logical applications. When dealing with problems that contain ambiguous information, conventional computers require long, complicated programs that are time consuming and costly to develop. NNs may use analog or digital signals. Performance is evaluated on the number of interconnections per second, instead of the number of instructions per second used to judge traditional computers. Effective management of large memories is more important than Central Processing Unit (CPU) speed. NNs attain their speed through massive parallelism. Individual analog neurons perform operations much more slowly than digital computers. Unlike ESs, NNs cannot explain how a particular conclusion was reached.

When designing an NN, decisions must be made about (Bowen 1991):

- The number of layers
- The number and type of neurons
- The size of the hidden layers
- The number and type of output neurons
- The connectivity of each neuron and layer

Mainframe computers are integrated widely into the NN infrastructure. In the near term, the use of conventional computing platforms are acceptable solutions for experimentation and prototyping, even though mainframe performance is less favorable than that of supercomputers and massively parallel machines. In the future, as more NN hardware is developed, NNs will most likely operate on workstations and personal computers (PCs).

2.4.7 Future of Neural Networks.

Table 2.4-1 illustrates some of the needs and future applications for NNs.

TABLE 2.4-1. FUTURE EXPECTATIONS FOR NEURAL NETWORKS

Needs
• More powerful serial digital computers
• Better understanding of brain operation
• Developments in hardware that will support NN speed and storage requirements
Applications
• NNs integrated with ESs
• Space exploration
• Nuclear power
• Military
• Applications where an NN's graceful degradation would allow operations to continue under circumstances in which conventional computers would cease operations
• Any applications that would benefit from the unique pattern recognition and identification capabilities of NNs

2.5 INTEGRATED AI SYSTEMS.

2.5.1 Fuzzy Logic Integrated with Neural Networks.

Integrating NNs with other technologies may have considerable potential. The difference between fuzzy and neural systems is in the way they represent knowledge. By adding fuzziness to an NN, the designer can describe boundaries more generally, describe input more vaguely, and maintain better control. Few systems are capable of creating fuzzy decision rules and modifying them on the basis of past experiences. The synthesis of fuzzy logic and NNs has control and image processing applications.

Adaptive fuzzy systems are much like NNs. Both are given a set of inputs and the desired outputs for training. NNs, however, cannot explain how a conclusion was reached.

Neuro-fuzzy products are becoming a popular option for applications such as image recognition. Neurons in fuzzy logic devices operate like other NNs, except that there are fuzzy sets as inputs and outputs. NNs can assist fuzzy systems in refining the degree of membership functions and the rule bases. They can also be used to optimize the parameters of a fuzzy logic system.

One such system may have an inference engine based on the principles of approximate reasoning and have the learning capabilities provided by neuron-like elements.

Fuzzy system knowledge is implanted by human experts. With NNs, knowledge is learned by the system.

2.5.2 Expert Systems Integrated with Neural Networks.

ESs are suited for problems that are well-defined and specific to a certain domain, while NNs can deal with a broader and more general range of problems. ES implementation tends to be a long process, while NNs can provide answers quickly and with acceptable accuracy.

The goal of integrating these two methods is to extract features from NNs and ESs to create a system that can provide the advantages of both. The advantages include reasoning for collecting data and fast, general answers or more accurate, slower ones. Also, ESs can analyze and validate an NN's output.

There are three steps involved in an ES/NN problem solving process: data collection, data evaluation, and conclusion analysis. Data collection and conclusion analysis are tasks best suited for the ES component. ESs are designed for user interaction, such as collecting inputs and producing outputs. Data evaluation is a task well-suited for the NN part of the system, because it deals with imprecision and learning.

Because ESs adjust to the skill level of the user, NN access is provided to a wider range of users. The time required to train the NNs can be reduced because there is no need to account for all possible combinations of a particular problem.

2.5.3 Expert Systems Integrated with Fuzzy Logic.

A fuzzy ES is an ES that incorporates fuzzy sets and fuzzy logic into its reasoning process and knowledge representation. This allows a wide range of problems to be solved with the benefits of ES techniques and fuzzy theory. Fuzzy ESs can be user friendly, efficient, and compact.

In some areas, fuzzy methods can replace traditional methods altogether. In some applications, fuzzy techniques can augment other methods.

Inference procedures that can handle imprecise or vague knowledge are beginning to be recognized as useful problem solving methods. When developing the rule base in an ES, experts often express their knowledge in vague terms. An ES can handle imprecision in controlled amounts. When problems have uncertainty, fuzzy logic is the desired choice of methods. It has been employed successfully in several ESs. Fuzzy ESs use human-like behavior to handle concepts and reasoning.

Fuzzy logic is especially helpful for managing the rule base of an ES. It is nearly impossible to account for all possible combinations of antecedents and conclusions for any particular problem, especially large, complex problems. The logic needs to be traced, step by step, from all starting points to possible solutions. This task can be nearly impossible when dealing with even medium-sized knowledge bases. Fuzzy concepts, however, allow all combinations to be covered using fewer rules. Some action always will be taken, even if an exact match of the antecedent is achieved (Togai and Watanabe 1992).

Many fuzzy systems that are currently implemented are integrated with ESs. Using a fuzzy approach to ESs greatly simplifies knowledge acquisition and representation (Williams 1992). There are many situations where an expert needs fuzzy terms for accurate descriptions. The rule base is determined from the expert's knowledge. In general, fuzzy logic is a mathematical foundation that has applications in many disciplines.

3. ARTIFICIAL INTELLIGENCE AND EXPERT SYSTEM DEVELOPMENT.

This section examines the software, hardware, and development environment related to the AI field. The software and hardware development tools and hardware devices discussed in this section are mentioned as a sampling of available tools, systems, and devices, and not intended to be a complete listing of what is available.

Complete solutions for AI-based problems are not available exclusively in silicon at this time. However, in the future this could change as technology continues to drive IC densities to greater levels of complexity and AI technology continues to mature. Whether it is a microprocessor with an AI-based native language, or an NN IC, the device is generally viewed as part of a larger system. Current applications for AI-based devices are proposed for automating and assisting human or machine performance. AI-based devices typically function as embedded processors.

Whether one is concerned only with hardware devices, or only with developing an AI-based software program, tools are required. Many issues should be examined when developers are considering a tool for building an AI-based system. There are many tools available, but the suitability of a tool should be examined carefully. Tools may be chosen by financial motivation, the developer's familiarity with the tool, or the tool's compatibility with the developer's hardware. In some cases, tools are developed and applications are then fabricated to test them (Waterman 1986).

Some basic guidelines for tool selection are listed below (Waterman 1986):

- The tool should provide the power and sophistication required by the development team.
- The tool's support facilities should be adequate for the development time frame.
- The tool should be reliable.
- The tool should be maintainable.
- The tool should have features to suit the needs of the problem being addressed.
- The tool should have features to suit the needs of the application being addressed.

3.1 SOFTWARE FOR ARTIFICIAL INTELLIGENCE APPLICATIONS.

There are many computer languages available for the program developer. Having a digital processor perform what the programmer desires has evolved from coding individual bits for each memory location to programming on a keyboard in a language similar to spoken English.

Programming large applications in assembly language (a low level language using mnemonics to represent the processor's machine code) is inefficient, costly, and difficult to debug since there is a large body of code to review. Higher level languages and other development tools have been created to allow the programmer to develop and debug code rapidly.

Along with the creation of languages and tools came increases in the complexity of digital systems and the corresponding complexity of programs required to control them. Languages created specifically for the purpose of programming AI-based systems include LISP and PROLOG. Other high level languages, such as those that follow, have been used as well:

- FORTRAN (Daysh, et al. 1991)

- FORTH (Williams and Davidson 1991)
- Ada (Leeper 1990)
- C (Masotto, Babikyan, and Harper 1990)
- C++ (Shewhart 1992)
- Pascal (White and Bart 1988)

Shaw^2 (1993) provides a list of AI language products and tools available, including LISP, Scheme, Rete, Logo, and PROLOG products. Some of the common AI-specific programming languages are presented briefly in the following sections.

3.1.1 LISt Processor.

LISP is called the native language of AI (Charniak and McDermott 1985). Many different versions of LISP have been developed. COMMON LISP was developed in an attempt to focus these versions in a common direction and enhance the language portability.

Users can change the syntax of the LISP language to suit themselves. It is symbol instead of number oriented. Data structures are easily constructed in LISP. LISP is also procedure oriented. Nested subroutines are used to organize and control program execution.

LISP was designed for manipulating symbols, as opposed to numeric data. The ability to manipulate symbols has made LISP popular among the research community. Programming in LISP provides certain benefits:

- An interactive programming environment
- Incremental compilation
- Customization based on the application
- Automatic memory management features

Programs in LISP are written by adding definitions of new functions and other program parts. The method of interaction is through a *read-eval-print* loop. Steps in this loop include reading the user input, converting it into a LISP expression, evaluating the expression, and then printing the results of the evaluation.

Being interactive, LISP is not burdened with the edit-compile-link-run cycles that are used in the conventional programming environment. LISP users are not faced with the task of writing a main program, along with the associated drivers. When a function is redefined, immediate results are produced. Also, with LISP, interpreted and compiled definitions generally can be mixed.

One of the features of LISP is the ability to allocate memory for data structures during "run time." When the data structures are no longer needed by the program, the system running LISP must do "garbage collection." Garbage collection returns the memory for system use. LISP has been primarily an interpreted, as opposed to a compiled, language. Statements are translated and executed one by one with an interpreted language. A compiled language is translated to machine code before it is executed. Where high performance is required, however, LISP must be compiled (Layer and Richardson 1991).

3.1.2 PROLOG.

PROLOG is a programming language based on mathematical logic. Finding a solution is a matter of deducing it from the facts. PROLOG has been successful in natural language processing applications and has been an important language base for the study of parallel programming techniques (Murphy 1993).

PROLOG software packages come in interpreted and compiled versions, and allow for the use of recursion. Programs consist of two main elements: predicates and rules. Predicates are used to express relationships between objects, as the following examples illustrate:

```
child_of(harry,sam).
child_of(harry,joan).
```

These two predicates declare that Sam is a child of Harry and that Joan is a child of Harry. Rules consist of a head and a body. The head is a predicate and the body consists of one or more rules. A sibling relationship is determined by the following rule:

```
sibling(sam,joan) :-
   child_of(harry,sam),
      child_of(harry,joan).
```

This rule declares a sibling relationship between Sam and Joan, if Sam is a child of Harry and Joan is a child of Harry.

While PROLOG is suited for handling logic such as this, it is not suited for number crunching or interfacing to real-time data collection systems (Murphy 1993). Programming is often performed using two languages to handle the required functionality efficiently. For instance, C may be combined with PROLOG to be effective in a program with a mixture of numeric computation and logic.

One example of the use of PROLOG is by Boeing Corporation in the Connector Assembly Specifications Expert (CASEy). CASEy is an ES that provides electrical process specifications for wiring harness assemblies. It generates a list of required tools, procedures, and material to perform the task. CASEy reduces the time for an operator to assemble an instruction set from 42 to 5 minutes (Roth 1993).

3.1.3 Tools for the Expert System.

ES shells are used to develop knowledge base systems. Two mentioned by Rouse (1990) are "Knowledge Engineering Environment" and "Acquisition of Strategic Knowledge."

An ES shell provides the human interface for programming an ES. Shells limit development flexibility, but save development time by providing a development environment specifically suited to ES programming. Shells provide a method for representing and storing domain knowledge in the form of rules and provide an inference engine for making decisions based on that knowledge.

For ESs, programming languages and ES shells are used for development. Table 3.1-1 (Waterman 1986) summarizes some of the advantages and disadvantages of using an ES shell, as opposed to a programming language, to implement an ES.

TABLE 3.1-1. COMPARISON OF EXPERT SYSTEM AND PROGRAMMING LANGUAGE DEVELOPMENT METHODS

	Expert Systems	Programming Languages
Advantages	Development is easier, faster, and, therefore, less costly	Greater flexibility
	More guidelines and mechanisms for representing and accessing the knowledge base	Product may more closely match the problem being solved
Disadvantages	Less flexibility	Developer will need to design the knowledge base and the inference engine
	May lack efficiency	Takes longer to develop application

Level5 Object® is an example of one of the many available ES development tools. Level5 Object is an object-oriented development tool running in a Windows® environment. Some of the features of this tool include a database interface using an object-oriented database management system and support for hypertext application development by the provision of some of the basic support tools required.

The user is given several methods of inferencing and control. The following techniques are used in Level5 Object (Level5 Object User's Guide 1992):

- Forward Chaining – Starts with known conditions and searches forward in the knowledge base to determine what can be concluded.

- Backward Chaining – Starts with a specific hypothesis or hypotheses (agenda). The inference engine then searches backward from the agenda using various search order strategies.

- Mixed Mode – Uses a combination of forward and backward chaining.

- Procedural – Allows the user to write rules or demons (a procedure invoked when data are changed or read), or utilize the Production Rule Language programming method supplied with the development package.

- Object-Oriented – This type of inferencing processes a knowledge base's methods, classes, instances, attributes, and facets.

- Point-to-Point Hypertext – Uses displays, hyperregions, push buttons, and demons to allow the user to move through the text.

Lischke and Meyer (1992) describe the Technical Expert Aircraft Maintenance System (TEAMS), an ES for aircraft maintenance that uses Level5 Object.

3.1.4 Fuzzy Languages for Development.

Language tools exist for generating assembly code for popular microprocessors based on a language translation from a predefined set of fuzzy instructions. These tools operate on the fuzzy instruction set and convert each instruction into a sequence of assembly code mnemonics for the selected microprocessor. Tools also exist for fuzzy processors with a fuzzy-based native language. These tools define a particular fuzzy assembly language that requires no conversion process. Instructions are converted from assembly code directly into executable machine code for the fuzzy controller.

Some tools also generate C code from the translated fuzzy instruction set. The C code can be compiled and executed on a number of processors. These tools allow the developer to program using a fuzzy command set. A translation is then performed to create high-level C code. The C code can be compiled and executed on numerous processors and systems.

3.1.5 Other Languages and Tools.

Procedures, standards, and development methodologies have been produced for the use of conventional programming languages in digital avionic systems, but do not exist for AI specific languages. Since this is so, some of the proposed avionic applications of AI-based systems have focused on the use of conventional software for the implementation. In the military community, the use of Ada has been mandated for future avionics and space applications.

A goal for large scale development of AI-based systems should be to automate as much of the process as possible by developing tools or by using tools that are available and suitable. Tools focused on automating the design process decrease the development cycle time. They also reduce the number of human errors.

Numerous specialized tools have been developed to facilitate the development of AI-based systems. Variations of LISP and PROLOG exist, in addition to numerous ES shells and other tools. A number of trade magazines and journals are published that identify available development tools and environments. Part of the problem for AI developers is not only to specify system functionality clearly, but also to identify the right tool or tools for the job. For instance, one magazine contains a listing of 74 available resources for the NN development environment. Listed are development tools and environments, development computers, and NN training software (Shaw[1] 1993).

3.1.5.1 Boeing Advanced BlackBoard Ada Generation Environment.

One of the tools being developed at Boeing is the Boeing Advanced BlackBoard Ada Generation Environment (BABBAGE). This tool is designed to be a complete work environment for the development of AI-based systems.

Leeper (1990) indicates that the need for such a tool set arose due to the following:

- Reluctance by AI programmers to deal with strongly "typed" languages, such as Ada
- Necessity to reduce syntactic and semantic coding errors
- Inflexibility of software interfaces to the AI engine
- Requirement to eliminate laxity in programming loosely typed languages

Figure 3.1-1 shows the BABBAGE tool elements and their relationships.

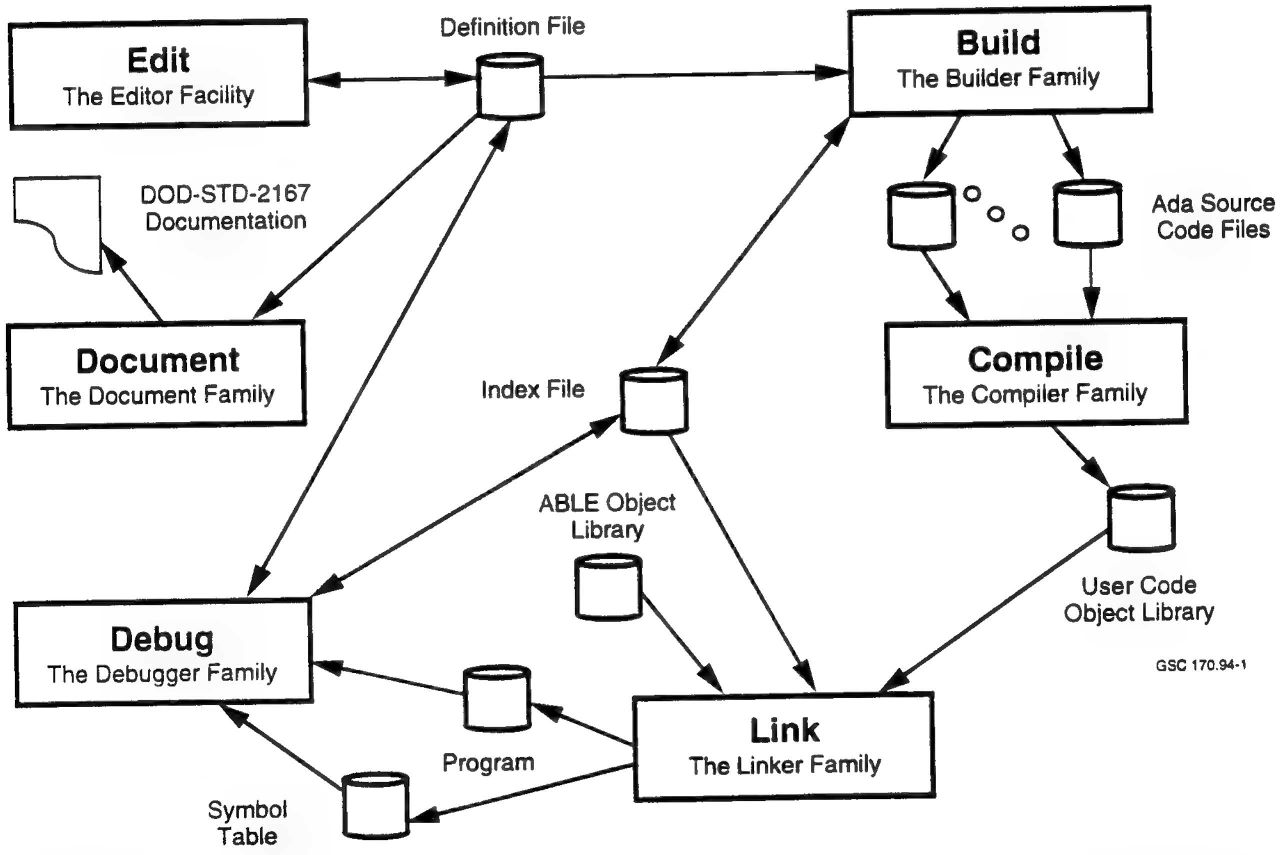


FIGURE 3.1-1. ELEMENTS OF THE BABBAGE TOOL AND THEIR RELATIONSHIPS
(Leeper 1990)

Structural aspects of the AI program are described in the Definition File, which is maintained by the editor facility. Basic editing services are provided by the EDIT facility, as well as other time-saving features, such as recognition of the syntax of the definition language. The BUILD facility provides the services required for translating the contents of the definition file into Ada source code files. It verifies syntax and semantics and generates appropriate error messages.

A COMPILE facility compiles the source code files generated by the BUILD facility. The index file is used to identify the files needing compilation and the correct compilation order. ABLE is an Ada software blackboard manager. The object code from ABLE and the other compiled code are used by LINK to create the executable file. A symbol table is generated for use by the DEBUG facility. The DEBUG facility provides the functions necessary for verifying correct operation of the program and capabilities to assist in identifying and locating software errors. A DOCUMENT facility is provided that allows the user to define the documentation level and provide documentation based on DOD-STD-2167A.

3.1.5.2 CLIPS Expert System Shell.

CLIPS is a knowledge base tool developed at NASA Johnson Space Flight Center. It was written in American National Standards Institute (ANSI) C. It is an ES shell that has a LISP-like syntax. The basic elements of CLIPS are as follows:

- A list of facts
- A knowledge base containing the rules
- An inference engine

The inference engine decides which rules to fire based on the facts. Rules fire if facts are asserted and are true.

Some of the features of CLIPS include (Daysh, et al. 1991):

- Use of variables for control of rules and facts
- Capability to perform LISP-like calculations within rules
- Ability to prioritize the rule firing sequence
- Provision of commands to aid in debugging

3.1.5.3 FORTRAN Library for Expert Systems.

The FORTRAN Library for EXpert systems (FLEX) was developed by the Royal Aerospace Establishment (RAE). It consists of a library of FORTRAN 77 subroutines that allows FORTRAN programs to interface with ESs. In addition to the subroutines, FLEX supports a separate knowledge base and provides explanation facilities (Daysh, et al. 1991). Butler and Corbin (1988) give further details on the features and operation of FLEX.

3.2 ARTIFICIAL INTELLIGENCE AND EXPERT SYSTEMS HARDWARE.

Development of AI applications based on LISP began on an IBM 709. A high-performance application, MacLisp, appeared on the PDP-10. Other special purpose LISP machines began to appear, such as the Xerox PARC and Symbolics systems. Now, systems implementing AI-based applications are being developed using generic systems with compilers for the particular task at hand. Development environments exist for mainframes, workstations, and PCs.

Target environments for implementations vary widely. For instance, one AI implementation was performed using a MIL-STD-1750A processor, 64k memory, and the Ada programming language.

Recently, however, great strides have been made in the development of silicon-based devices that implement specific AI technologies. Most notable are high-speed and dense NNs and very fast fuzzy logic processors.

3.2.1 Neural Processors.

NNs are implemented in both software and hardware. A software-implemented NN may be a standalone or an add-on software package which can run on a number of computer platforms. Inputs to the NN, as well as the weights, are stored in computer memory. The software performs processing based on a fixed or user-selectable algorithm. The output state is then determined based on the inputs, NN weights, and calculation method. Typically, users adjust the weights, choose the calculation method, and train the network. Software-based NNs may derive inputs from various external or internal sources, such as spreadsheets, databases, and custom input files. Where higher levels of performance are necessary, NNs are implemented using hardware accelerators or NN ICs. Neural processors have been the focus of significant research recently. The following sections examine some of the silicon-based AI hardware that is either currently available or in some phase of research and development.

A number of applications are suited to the use of neural processors. Some of these applications include target recognition, sensor fusion, sonar signature identification, voice recognition, and unmanned vehicle control. In these applications, the neural processor has the distinct advantage of being much faster than a Von Neuman-based CPU, which executes instructions sequentially. A neural processor is a special function parallel processor, and, hence, is much faster in the execution of these special functions. In addition, it can be retrained rapidly when compared to rewrites of software algorithms for sequential CPUs.

3.2.1.1 80170NX Electrically Trainable Analog Neural Network.

The 80170NX Electrically Trainable Analog Neural Network (ETANN) is an Electrically Erasable Programmable Read Only Memory (EEPROM)-based IC available in a 5-volt pin grid array (PGA) package with 208 pins. It is both Transistor-Transistor Logic (TTL) and Complimentary Metal-Oxide Semiconductor (CMOS) compatible at the interface connections (Intel[1] 1990).

The ETANN is internally configured with 64 neurons and 10,240 synapses. Inputs to the ETANN are both internal (feedback) and external. The device feedback inputs can also be used as device inputs, providing up to 128 total inputs. Performance claimed for this device is greater than 2×10^9 connections per second.

The use of EEPROM makes the device weights reprogrammable and nonvolatile. Individual weights and neurons are addressed using the device's address lines, and the weights are stored internally. A Learn Control pin allows the NN to operate in a learning mode, which causes the values of the synapse weights to change.

The typical internal connections of the ETANN device are given in figure 3.2-1.

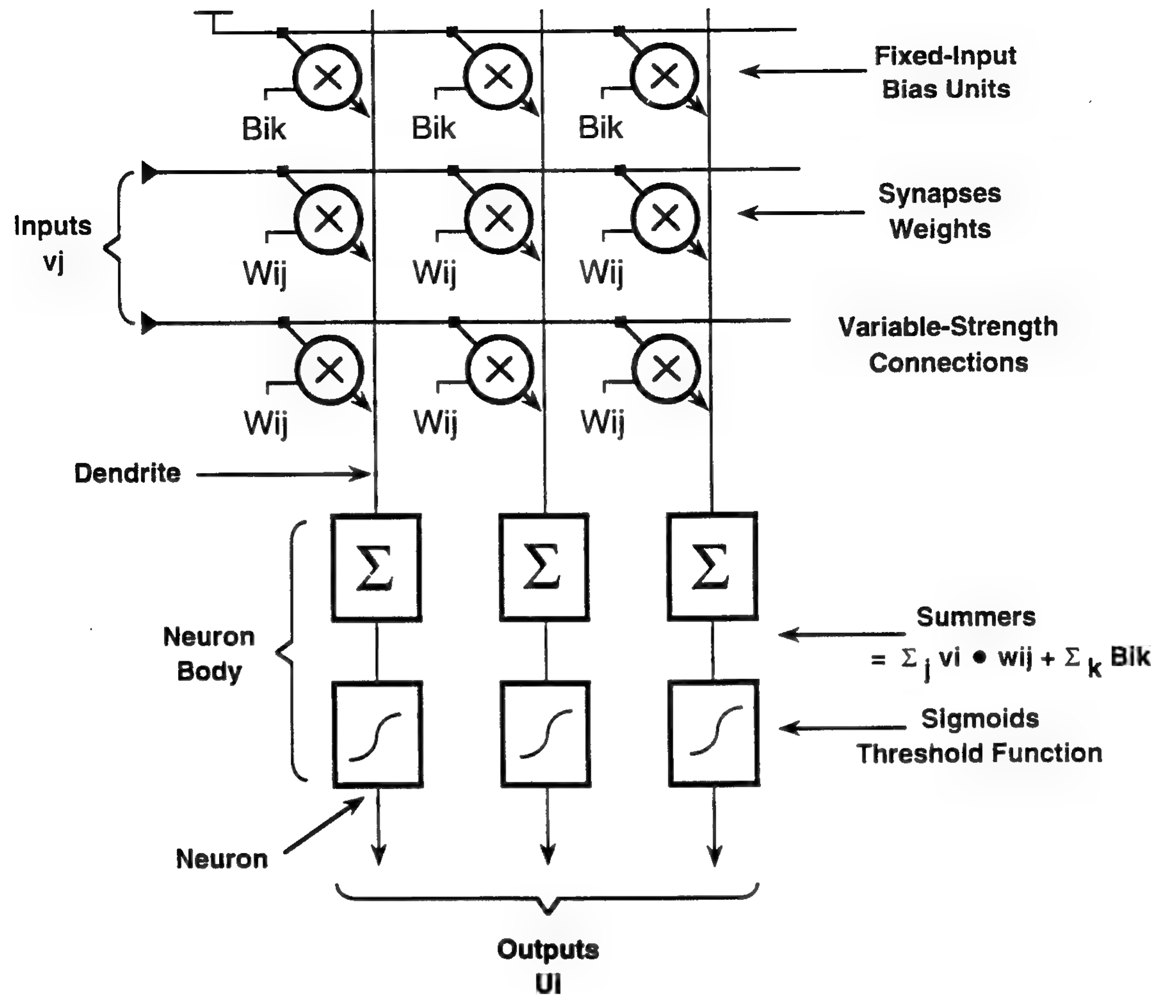


FIGURE 3.2-1. REPRESENTATIVE CONNECTIONS FOR THE ELECTRICALLY TRAINABLE ANALOG NEURAL NETWORK (Intel[1] 1990)

The output is calculated as follows:

$$u_i = \text{Sigmoid}\ \{\Sigma_j\ \text{Weight}(w_{ij})\bullet\text{Input}(v_j) + \Sigma_k\ \text{Bias}_{ik}\}$$

where:

u_i represents an individual neuron's output,
w_{ij} represents the user-programmed weights of the ETANN,
v_j represents the value of each input, and
Bias_{ik}, associated with each output, allows for control of the neuron threshold.

Both w_{ij} and v_j represent vectors which have both magnitude (voltage) and direction (polarity). A dot product between these two vectors produces a scalar value representing the difference between the weight and input values. When the angle between the vectors (polarity) is the same, then the largest dot product is produced.

A summation of these products is formed and the associated bias is added. This total is then applied to the threshold function which determines the output state for that particular neuron.

NNs are typically made for cascade operation. The outputs and inputs are voltage and current compatible so that the output from the first device connects directly to the input of the second device. Connections between ETANN devices can be made using either a bus interconnection scheme or direct pin-to-pin wiring (Intel[1] 1990).

3.2.1.2 Ni1000 Neural Network Integrated Circuit.

One of the programs of the Advanced Research Projects Agency (ARPA [formerly DARPA]) is the Artificial Neural Network Technology program. One of the directions of this program is to fund development and applications of the Ni1000. Development is being performed by Intel and Nestor. Lockheed Missiles and Space Company is the first Beta test site for this device (Coleman 1993).

The Ni1000 design contains 1024 artificial neurons, implemented using 3.7 million transistors. Processing speed for the device is claimed to be 20 billion integer operations per second. The IC uses a standard microprocessor bus interface. Targeted applications include sensor fusion, sonar signature identification, voice and target recognition, unmanned vehicle control, and real-time fault monitoring and diagnosis.

3.2.1.3 RN-200 Neurocomputer.

This Neurocomputer IC was designed by Ricoh of Menlo Park, California. It is fabricated using CMOS technology as a gate array device with 200,000 gates, using .8-μm line widths. A total of 256 synapses are implemented on the IC. It consists of three layers: the input, processor, and output layers. The neuron update rate is claimed to be 1.5 billion per second when the device is running at a processing speed of 12 MHz (Normile 1992).

3.2.1.4 M1718 Digital Neural Network.

The M1718 Digital Neural Network is produced by Hughes Semiconductor. This device utilizes static CMOS memory internally. It contains 1024 weights, which support 32 8-bit inputs connected to 32 internal nodes. The M1718 is TTL-compatible with an 8-bit bus interface. A throughput of 100,000 patterns per second is claimed by the manufacturer. The devices are gangable so that networks of varying sizes can be created.

3.2.1.5 Optical Neural Computing Integrated Circuit.

A prototype optical neural computing IC has been produced (Mitsubishi Electric 1992). The design, shown in figure 3.2-2, uses eight long and narrow Light Emitting Diodes (LEDs) on top of an 8 x 8 array of photodetectors. Weighting is performed by adjusting the sensitivity of the detectors. This IC is fabricated on a gallium arsenide (GaAs) substrate that measures 6 mm square. It learns at a rate of 600 million connection updates per second.

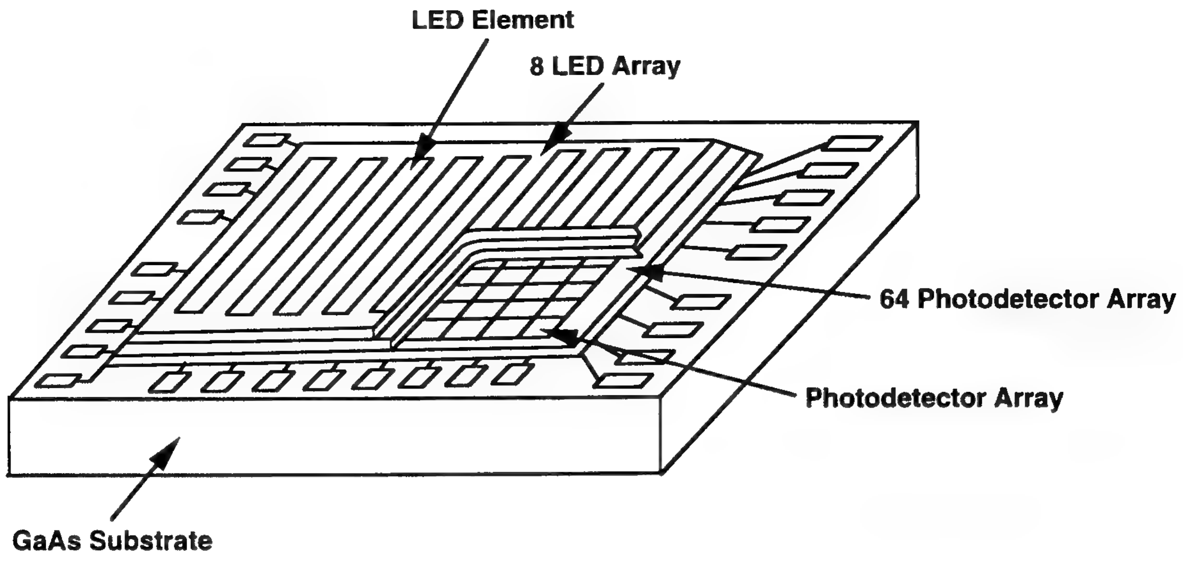


FIGURE 3.2-2. OPTICAL NEURAL COMPUTING INTEGRATED CIRCUIT
(Mitsubishi 1992)

3.2.1.6 Neural Accelerator Hardware.

Neural accelerator hardware implements an NN on a printed circuit card that is hosted on a computer system such as a PC. Neural accelerators use special high-speed processors, architecture, and other supporting hardware to attain fast connection rates, as compared to software-implemented or NN ICs.

One neural accelerator uses the XP50 and XR25 RISC (Reduced Instruction Set Computer) processors. The XP50 is a 100 MFLOPS (Million Floating Point Operations per Second) device. The manufacturer claims that connection rates of up to 45 million per second can be attained.

Since these cards are based on RISC processors, they require both hardware and software support. Large amounts of Random Access Memory are generally utilized in such applications. One system utilizes up to 64 megabytes (NeuroDynamx 1994). Development packages may include libraries with special functions, interface drivers, graphics, and compilers.

3.2.2 Design Considerations for Fuzzy Logic.

In cases where a fuzzy logic solution matches the intended application, designers can achieve significant advantages. Using fuzzy logic, system designers can realize lower development costs, superior features, and better end product performance. Products can be brought to market faster and more cost-effectively. Factors that make fuzzy logic desirable to the system designer include the following:

- Description and modeling solutions to a problem without complex mathematical models for systems and development.

- Optimization of known solutions to obtain a simpler and more effective implementation.

- Simplification of the system design process and corresponding decrease in development costs.
- Creation of a more descriptive system. A fuzzy logic-based system is more convenient to manage, maintain, and upgrade.
- Greater fault-tolerance and better trade-off between system robustness and system sensitivity.
- Provision of products with powerful features and performance within a price range unmatched by other solutions.

There are considerations to be made in the choice of a CPU architecture for a fuzzy logic-based system. The proposed application can dictate the hardware required. Conventional CPUs are sometimes used where execution speed is not a factor. Where speed is a factor, the choice of the CPU can have significant impact on system performance.

Architectures having compare instructions with three operands are beneficial since these instructions allow nondestructive compares. For example, a two-operand OR instruction is typically of the following form:

OR *destination, source*

When this instruction is executed, the destination is modified when the results of the OR operation are placed in it. The use of a third operand is necessary for a nondestructive compare.

A CPU architecture containing a large register file is helpful. This is necessary if all terms are to exist within the CPU. A CPU containing a large register file greatly reduces the number of external memory accesses that are required during program execution.

Applications vary widely with respect to the number of rules required and the speed of the CPU which executes those rules. Some speed-critical applications may require only a few rules, but a very fast CPU to keep in time with the controlled system. A high-speed motor control application is one example where speed is essential, but the number of rules applied can be limited.

Automotive applications require high speed and can be complex. Antilock braking systems, climate control, and transmission control are examples of applications that may be suited to the use of fuzzy logic. High-speed applications will require the use of special fuzzy logic ICs. These ICs may be fast enough to control several tasks simultaneously. Fuzzy controllers execute rules 10 to 100 times faster than general purpose CPUs (Legg 1993).

3.2.2.1 Fuzzy Coprocessors.

Fuzzy Coprocessors and "engines" are also produced by a number of manufacturers. Fuzzy Coprocessors are intended to interface to various microprocessors so that fuzzy instructions can be performed rapidly, by a device tailored for them. They execute instructions specifically targeted for them out of the CPU instruction stream. Fuzzy engines are designed to be used as a peripheral device within a CPU, or as a basic building block for an Application Specific Integrated Circuit (ASIC). Figure 3.2-3 shows some of the possible combinations developers can implement using ASIC core technology.

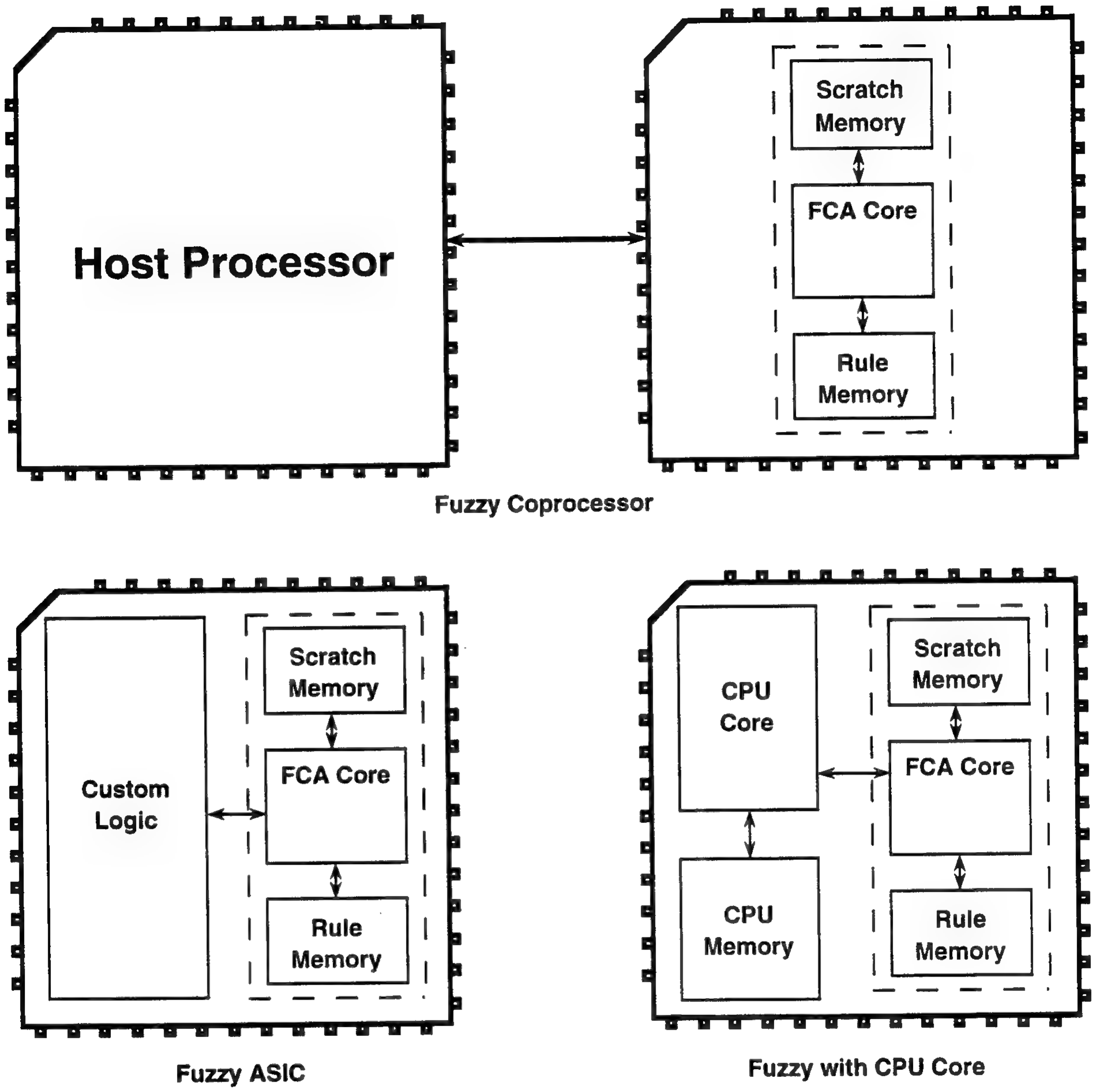


FIGURE 3.2-3. BASIC IMPLEMENTATIONS OF ASIC CORE USING FUZZY LOGIC
(The Fuzzy Source 1992)

These combinations include the Fuzzy Computational Accelerator (FCA) core. The ASIC elements can be combined to form various designs:

- ASICs with fuzzy capability and custom logic
- ASICs that combine fuzzy processing on the same IC as a conventional CPU
- ASICs that are designed to function as coprocessors

Some of the fuzzy engines and coprocessors that are under development or are in production are discussed in the following sections.

3.2.2.1.1 FC110 Fuzzy Processor.

The FC110 Fuzzy Processor was developed by Togai InfraLogic, Incorporated. The FC110 is called an "8-bit RISC processor optimized for fuzzy logic." It is designed for processing large rule bases and allows user-defined membership functions (see section 2.3) with arbitrary shapes (Legg 1993).

3.2.2.1.2 SAE 81C99 Coprocessor.

The SAE 81C99 coprocessor was developed by Siemens Corporation. Its features include an 8-bit interface to a microcontroller. Peak performance of this device is given at 7.9 million rules per second at a clock speed of 20 MHz. It has 256 inputs, 64 outputs, and handles up to 16,384 rules (Shear 1993).

3.2.2.1.3 VY86C570 Fuzzy Computational Accelerator.

The VY86C570 FCA was developed by VLSI Technology. It can execute in excess of 850,000 fuzzy rule evaluations per second when using two inputs, one output, and 40 rules. It is built around the Togai InfraLogic VY86C500 fuzzy processor core.

This device has a 12-bit data path, 256 x 12-bit data registers, and addresses up to 64k x 12 bits of external rule memory. It also can be used as a core design for custom ICs where fuzzy computational capabilities are desired. The VY86C570A has built-in self-test capability (Weiss 1993).

3.2.2.1.4 NLX230 Fuzzy Microcontroller.

The NLX230 fuzzy microcontroller is produced by American NeuraLogix. It is fabricated using 1.25-micron CMOS technology and can execute up to 30 million rules per second. The rule memory is 24 bits wide and can be programmed with up to 64 rules which are shared among the outputs (Ziemacki 1993).

3.2.2.1.5 NLX110 Fuzzy Pattern Comparator.

The NLX110 fuzzy pattern comparator from American NeuraLogix is a logic pattern comparator designed for pattern recognition of items, such as currency or voice and fingerprints. The device contains a built-in NN. The data rate of serial input patterns is up to 20 MHz.

The data are formatted into fields which are then compared against stored patterns within the IC. For each pattern, the differences are summed. The user defines the thresholds for exact, closest, and worst-match, or allows the IC to update them automatically based on successive comparisons (Frueh 1993).

3.3 ARTIFICIAL INTELLIGENCE DEVELOPMENT SYSTEMS AND TOOLS.

For large software programs to be manageable, development tools are essential. This is true whether the software is conventional or AI-based. Tools are beneficial for the following reasons:

- They increase productivity.
- Through their use, certain development steps can be automated.
- They can provide system or partial simulation.
- They can improve testing.

The time from conceptual design to testing and implementation can be reduced dramatically with the use of the correct tools. Many AI-based applications exist on a variety of computer systems.

3.3.1 Fuzzy Development Systems.

Systems developed using fuzzy logic concepts have used both conventional CPUs and special purpose fuzzy logic processors. For many applications, the general purpose CPU will suffice for the task at hand. For most current fuzzy logic applications, an 8-bit CPU is adequate. In more demanding applications, the use of a fuzzy logic processor is required (Legg 1993).

For developers of fuzzy systems, a number of tools are available:

- Assembly language code on a general purpose CPU
- Special fuzzy assembly code for fuzzy controllers
- C code generating tools
- Complete development environments

Speed requirements and system complexity dictate the hardware that is required. The use of a high level language, such as C, will slow down the execution speed when compared to a system developed using assembly code. For simpler applications, fuzzy controllers are not required and the system may be implemented by the use of look-up tables. Where rule execution speed is the primary requirement, high-speed fuzzy controllers are required. Between these two extremes, developers use general purpose CPUs with fuzzy logic software (Legg 1993). Figure 3.3-1 shows the relationship between the speed in rules per second and the CPU resolution.

The number of degrees of membership (see section 2.3) can be the determining factor when CPU resolution is considered. Complexity of the system (number of rules) is a determining factor when speed is considered.

As with any digital system development, tools should be a major consideration for the choice of IC used. Tools vary in capability, price, and user-friendliness. Tools are used not only in the development phase, but in the debug and testing phases as well. Some of the advantages tools can provide include (Legg 1993):

- Graphical interfaces to simplify editing and creation of membership functions
- Graphical display of controlled surface
- Tuning of a system while it is active

Tuning a fuzzy system empirically does not provide enough assurance of system stability. Research is addressing this issue.

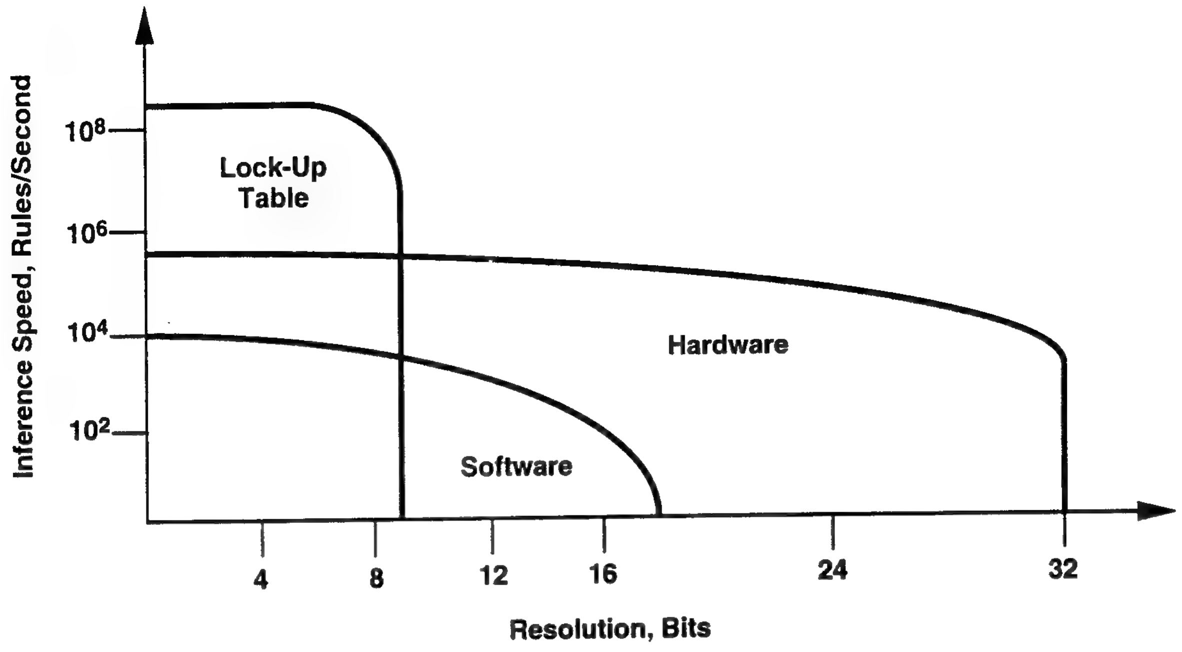


FIGURE 3.3-1. INFERENCE SPEED VERSUS CPU RESOLUTION (Legg 1993)

Development environments are available for fuzzy logic-based systems and provide system engineers with necessary tools. Tools exist for each level of debugging, including data tracing, analysis, and simulation. If the inference unit does not perform as expected, these tools can assist the user in locating bugs in the source code.

Different target processors are supported by a number of development environments. Some development environments contain a full compliment of tools to take the design from inception to completion, and are combined into a single software product. These are referred to as Integrated Development Environments.

As growth in this field continues, standard fuzzy language code and portability among the various processors will become issues. Tools that provide users with an open architecture environment allow flexibility in their use. Tools can provide the user with data formats for membership functions, inference rules, inference units, and other fuzzy inference parameters.

Key tools for development of fuzzy systems include the following:

- Editing Capability for Source Code Generation – Allows the user to specify I/O variables, the membership functions associated with them, and the inference rules. If the system requires special logic operators and inference methods, this capability needs to be included, in addition to any other specific requirements for tailoring source code.

  Graphic editors allow the user to graph the membership function, as opposed to using a text editor to describe a membership function.

- Compiler – The editor converts the graph into a segment of source code that is in the correct form for the compiler. The compiler converts the source code into a form that is executable on the specified processor.

- Simulation – This capability allows designers to simulate and perfect an entire system, including hardware, before building any hardware. Designs can be perfected, examined for trade-offs, and used to avoid costly errors in the fabrication or assembly stages.

  The simulator is also one of the final debug tools. It runs the simulation with input values prepared in an input value file. The simulator may display various parameter curves and output values as it simulates the unit's dynamic behavior. Integrated development tools may include links to the source code for additional debugging capability and simulation understanding.

- Debug Facility – One of the methods used for debugging is to employ a trace program. Designers can set input variables to any test value and observe the output. If undesired results occur, the inference process can be traced, step by step. This allows identification of the specific location where the source code bug exists.

Some tools may provide the user with a display of the global view of the transfer function response. I/O relationships are displayed as a 3-D structure. Interactive 3-D graphics allow users to check the function in detail, examining the surface from a variety of perspectives.

3.3.1.1 TILShell®.

One development environment available to designers is a graphical, object-based environment called TILShell®. This shell allows the user to define the production rules and membership functions. Three compilers are provided, depending on the choice of target system. These compilers allow code to be produced as C source code, microprocessor specific code, or machine code for the FC110 Digital Fuzzy Processor (Conner 1993).

3.3.1.2 Fuzzy Inference Development Environment.

The Fuzzy Inference Development Environment (FIDE) is a tool based on Fuzzy Inference Language (FIL). FIL is a non-sequential language with English-like statements that incorporates inference methods and a variety of logic operators for different applications. FIL also provides optimized flexibility for membership function representation and the data types required by separate target processors.

A complete standard for fuzzy systems is combined into one unit called the Fuzzy System Standard Environment (FSSE). To provide an open architecture environment, FSSE provides users with data formats for membership functions, inference rules, inference units, and other fuzzy inference parameters.

All source code is written in FIL. The user specifies the I/O variables, the membership functions associated with them, and the inference rules using FIL. If the system requires special logic operators and inference methods, FIL provides the capability to tailor the source code for specific requirements.

FIDE also includes a graphic editor for describing membership functions. The editor converts the graph into a segment of code. The FIDE compiler translates the source code into the standard data structure.

Debug tools, such as a trace facility, graphical analyzer, and simulation, are also provided. Once a processor has been selected for implementation, the corresponding real-time code generator can be used for that processor. If the unit is implemented in software, the run-time library, when linked into a C program, performs the fuzzy inference computation (Aptronix[1] 1992).

3.3.2 Fuzzy Logic for Digital Signal Processors.

Systems can be designed using a fuzzy development package built around the Digital Signal Processor (DSP). This allows the features of the fuzzy system environment to be utilized in the DSP field for greater flexibility in particular applications. The development software combines fuzzy logic programming and allows DSP techniques to be programmed for the same hardware. Such a tool is available for the PC environment (Texas Instruments Literature 1994).

3.3.3 Neural Network Development Tools.

NNs typically are not standalone devices. They generally are used as part of a larger system. A process passes data to the NN inputs. The NN then performs its operation and passes the output data back to the process. This is true for both software- and hardware-based NNs.

Designers need to keep in mind some potential drawbacks when using NNs.

There may not exist a satisfactory solution for a given problem if data are insufficient or if there is no learnable function. The problem must be definable. Training data need to be available and sufficient for the defined problem.

Proving consistency, completeness, and correctness for an NN may be difficult. The outputs of NNs depend on a high number of calculations. These calculations are based on input patterns and weights. The origin of the weights may be difficult to explain, since they are the result of a complex machine learning procedure.

NNs can be expensive to train. The training expense is due to the need to collect, study, and exercise the data. Additionally, the developer will need to manipulate parameters until the NN response is acceptable.

For NNs implemented using microprocessors, small increases in the number of nodes required cause large increases in execution time. As a rule of thumb, the execution time is roughly the square of the number of nodes. A microprocessor that works for a small number of nodes may not meet the required response time where the number of nodes is larger (Hammerstrom 1993).

Developers of NNs have the option of using a software-implemented NN or an NN hardware IC. Hardware devices are a relatively recent development and are generally expensive. However, if a software solution is too slow, a hardware NN may be required.

NN developers need to weigh the characteristics, requirements, and drawbacks of this technology carefully. Computational needs should be considered, including system throughput and the availability of training data. Sufficient data must be available to give a good representation of the behavior to be modeled.

If NNs are to be used in critical applications, other factors need to be considered. Hammerstrom (1993) sums up the design concerns effectively when he states:

> They are like statistics in their aggregate behavior, their usefulness, and their effectiveness in applications that lack other solutions and tolerate imprecision. They may be unsuitable when safety is critical or risks are to be avoided, unless they can be validated with all possible input values.

A number of manufacturers are marketing tools to assist in the design of NNs. NN development tools support software driven NNs and hardware NNs. Based on the design considerations, the developer chooses one of the two methods.

For the software-based NN, a variety of tools are available. Some of the tools interface with spreadsheet files for data. Different algorithms are offered, with most software tools allowing the developer to use any of several NN algorithms.

Two available tools for NN development are ExploreNet and KnowledgeNet. These tools assist the developer in applying NN techniques to solve engineering problems. The tools were designed to handle large data set analysis and pattern recognition, giving improved results over traditional analysis techniques (Aviation Week & Space Technology 1991).

An application for ExploreNet would be to predict aerodynamic performance of an aircraft. Input data for the NN can be gathered from finite element analysis, flight tests, and wind tunnel experiments. KnowledgeNet can be used to assist pilots in understanding why an engagement was won or lost. It was originally developed to assist in explaining decision rationale related to consumer loans (Aviation Week & Space Technology 1991).

Tools also exist for NN hardware development. Manufacturers that have produced NN ICs also support designers with complete development systems for those ICs. Some NN development systems use DSPs as hardware accelerators to allow processing speeds much greater than the host processor can handle.

Tools run on various systems, including DOS, Macintosh, Unix, VMS, and OS/2. Hammerstrom (1993) lists a variety of currently available development tools for both software- and hardware-based NN design.

3.3.4 Neural Network and Fuzzy Logic Combined System.

An NN and fuzzy logic system have been combined in a product called NeuFuz4, which generates code for the COP8 μC (Shear 1993). The development package uses an NN to assist in building fuzzy logic applications, since generating membership functions and fuzzy rules can be difficult.

Input and output data and other application parameters are used to create the fuzzy logic rules and membership sets. NN weights are adjusted until the outputs of the network closely match the outputs of the training set, given the same set of inputs. The final weights represent behavior that can be translated into rules and membership functions (Legg 1993). The rules are then examined by a rule verifier and processed by an optimizer. Figure 3.3-2 shows the functional blocks for this tool.

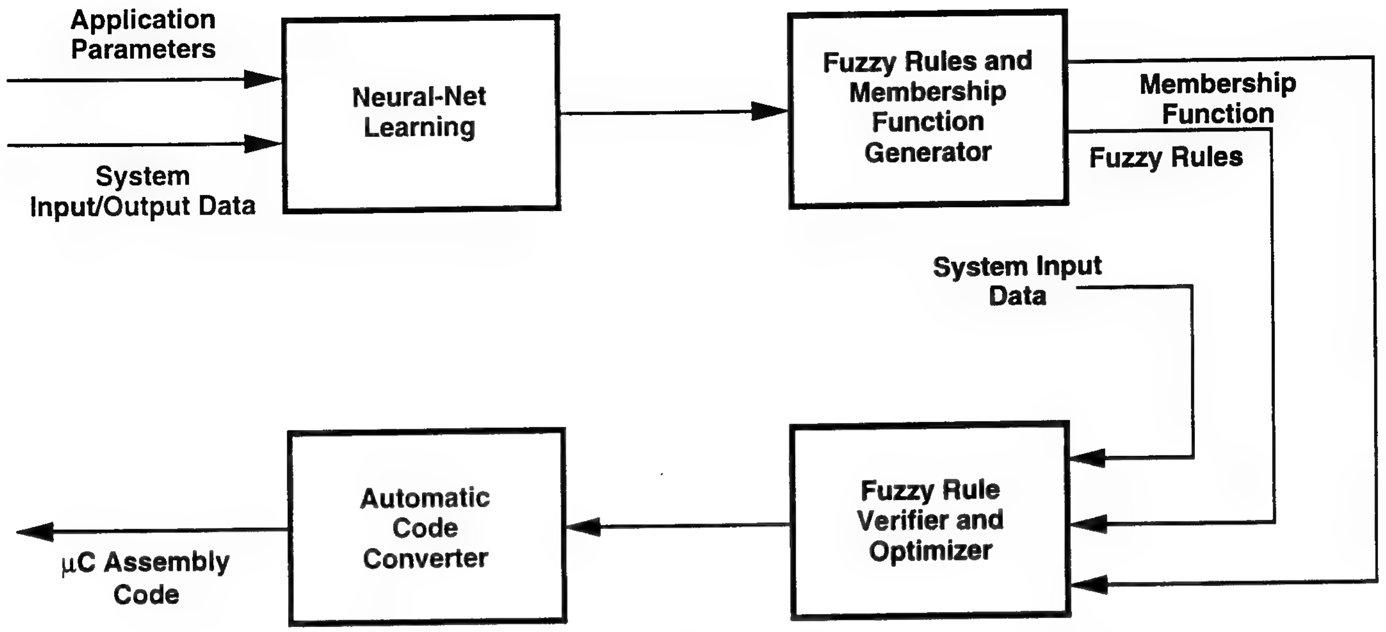


FIGURE 3.3-2. DEVELOPMENT TOOL BLOCK DIAGRAM
(Shear 1993)

When the developer is satisfied with the system simulation provided by the tool, machine code for the host processor can be generated.

The NeuFuz4 neural tool uses supervised learning. Also under development is a similar tool that uses unsupervised learning. It is anticipated that NN development tools used in training the fuzzy system will migrate into the fuzzy controller (Legg 1993).

4. ARTIFICIAL INTELLIGENCE AND EXPERT SYSTEM APPLICATIONS.

Much research has been performed in the field of AI. Numerous studies have been funded or performed by government laboratories and agencies, as well as private industry and universities. This section examines some of the aviation-related studies, particularly systems proposed for airborne application. AI-based aircraft monitoring and diagnostic systems are also examined. This section will identify some of the design considerations for such applications.

4.1 DESIGN CONSIDERATIONS FOR AVIONIC APPLICATIONS.

Designers of ESs must answer many questions during the course of development. These questions consider high level issues, such as system requirements, and black box issues, such as the type of system, inputs, processes, outputs, and interfaces. Many minute details must also be examined to implement the system.

4.1.1 Architecture Considerations.

Avionic subsystems for advanced military aircraft will provide a host of complex functionality, including the following:

- Terrain Following
- Threat Avoidance
- Weapons Delivery
- All Weather Operation
- Mission Planning

Within recent years, a number of AI-based applications have been proposed for use in civil transport aircraft. The proposed applications generally seek to enhance the safety or economics of flight. Following are some of the applications that have been proposed and studied:

- Systems designed to assist with emergency procedures
- Systems to provide navigation support
- Systems to provide support for diversions
- Systems to provide diagnostic and monitoring support

The high number of sensors on modern aircraft requires that data management be given careful consideration. Assimilation of the wealth of data available presents a difficult task for any crew. Information needs to be compressed and sorted, presenting meaningful and necessary results to the crew. This function will be computationally intensive, requiring reliable and high-throughput processing. AI-based systems are targeted to help reduce the amount of information requiring the pilot's attention.

When designing an ES, one must choose an architecture consistent with the requirements. The influence of timing considerations on the architecture and hardware selected for the development task will need to be determined. The system timing requirements and architecture choice influence the implementation. Designers are faced with many decisions, including whether to choose a single CPU or a multi-processor configuration; conventional, RISC, or special purpose CPUs; message passing schemes; memory design; operating system; and numerous other related issues.

One example of a high level architecture design is found in Masotto, Babikyan, and Harper (1990). This architecture was developed under a NASA contract and was suggested for use in the Adaptive Tactical Navigator program. One of the research goals was to "host and execute an intelligent system on multiple processors of the Fault Tolerant Parallel Processor (FTPP)." The associated report defines the architecture, functions, and design considerations. Figure 4.1-1 depicts a portion of the system architecture.

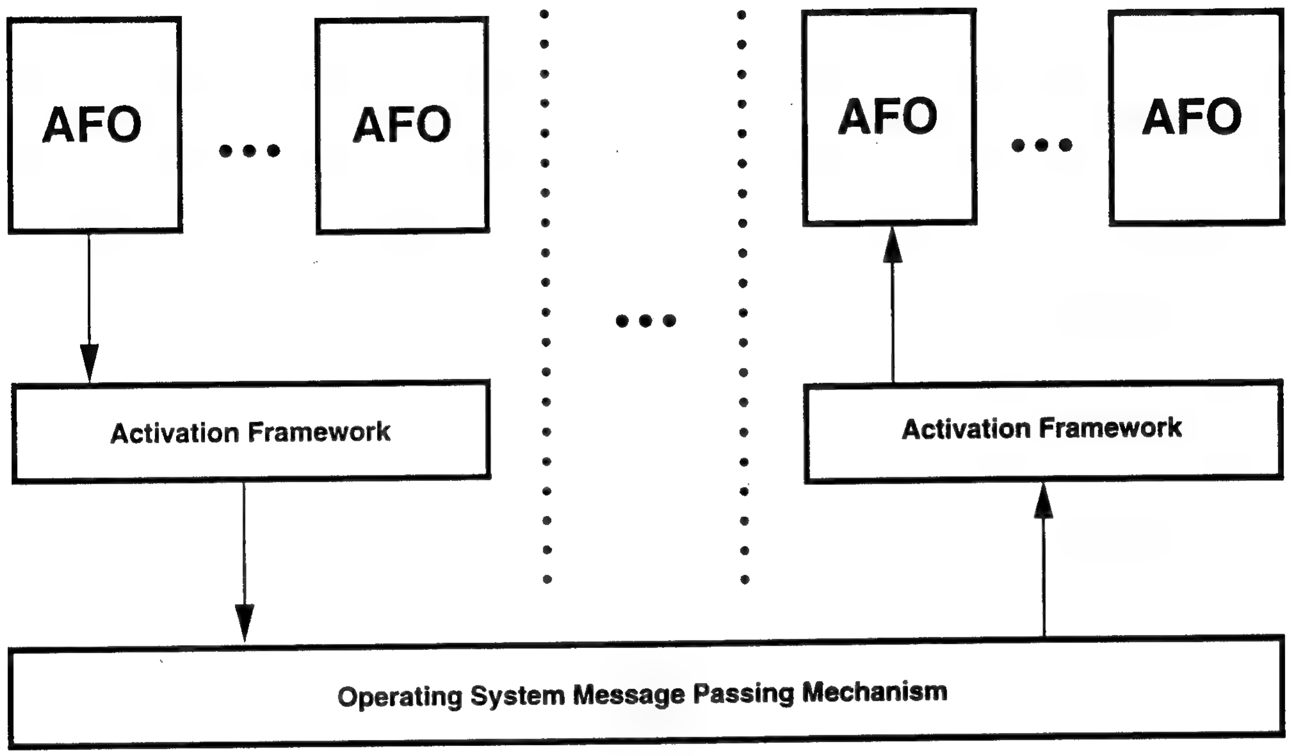


FIGURE 4.1-1. FTPP ARCHITECTURE
(Masotto, Babikyan, and Harper 1990)

The Activation Framework (AF) initiates and schedules the Activation Framework Objects (AFOs). The AFOs model the intelligence of the desired application. The AFOs are scheduled by the AF, in order of importance and priming conditions. An AFO is classified in importance by a unique priority number assigned to it. Priming conditions must be met before the AFO fires. The presence of messages at specific input ports is used to meet the priming conditions. Once the AFO is primed, it is scheduled for execution. Execution continues until it is preempted by a higher priority AFO or the task comes to completion.

4.1.2 Timing Considerations.

Timing is another example of a necessary, but important, detail when designing an ES for quick response or real-time operation. Following are some of the timing issues designers need to be aware of, or implement in their ESs, according to Ng (1988):

- Demonstrating that under any given scenario the system will provide data when desired, in a timely manner.
- Addressing response latency. Algorithms, such as those that perform LISP garbage collection, should be formulated with a time-sliced version to reduce response latency.
- Bounding problems so that the best reasonable solution under the circumstances can be identified.
- Eliminating unnecessary overhead from interactive development environments.
- Matching the processor architecture to the High-Level Language.
- Where higher speeds are required, using parallel algorithms, RISC processors, or other special hardware.

4.2 FLIGHT MANAGEMENT EXPERT SYSTEM.

As aircraft systems become increasingly complex, the pilot's mental and physical workload will increase beyond realistic limits. Expert flight management systems can assist decision-making processes for the pilot. However, problems can arise in acquiring, structuring, and applying the knowledge base needed by an expert flight management system capable of operating in a dynamic domain.

The Emergency Procedures Expert System (EPES) is a decision aid system suited for flight management applications. The goal of the system is to provide pilot assistance during emergency situations while keeping the pilot in the loop. During normal operations, the EPES would remain unseen to the pilot, but would monitor the state of the aircraft constantly. If an emergency were to arise, the EPES would initiate steps to correct the emergency, inform the pilot of the action, and allow the pilot to override the action, if desired.

The EPES is comprised of an environmental simulation and an ES. The environmental simulation models the behavior of the aircraft and provides a display panel with realistic dials and gauges. The ES contains a special purpose inference engine and the knowledge base.

A rule/goal paradigm for managing emergencies forms the basis for the knowledge base. Data-driven, condition-action rules are triggered by changes in the state of the knowledge base. Corrective actions are defined by goals that become active when one or more emergencies arise.

The EPES is intended to represent the knowledge of an experienced pilot in handling multiple emergencies. A top-down approach to knowledge acquisition was used in the development process. The knowledge base was developed incrementally over time at increasing levels of detail. The knowledge base was developed as a collaboration between the knowledge engineer and pilot. The knowledge engineer gathered basic emergency information from the flight manual. However, to create a system that performed at an expert level, the knowledge of an experienced pilot was needed.

Before the initial interview with the pilot, the knowledge engineer formulated a problem definition. During the interview, the knowledge engineer and the pilot confirmed their mutual understanding of the problem definition. The problem domain was divided into subproblems that were assigned priorities based on the pilot's knowledge. It was found that the domain readily divided into individual emergency procedures. These were discussed with the pilot at later interviews.

The knowledge engineer followed a pre-written agenda of questions. A general-to-specific approach was taken. This approach led to increasingly detailed questions and discussion to confirm the procedures required to recover from the emergency. Care was taken to ensure that knowledge was represented accurately. When the pilot explained a portion of an emergency procedure, the knowledge engineer recorded it and repeated it back. The accuracy of the knowledge was verified. Discussion continued until an understanding of the emergency procedure was reached.

After the initial interview for a particular emergency procedure, the knowledge engineer converted notes into a narrative explanation of the reasoning process. The narrative description was reviewed to confirm its accuracy. After the knowledge engineer and the pilot approved the content of the description, the knowledge engineer structured the information into two types of knowledge: diagnostic and remedial. Diagnostic knowledge detects changes in the aircraft and tests the current state of the aircraft. Remedial knowledge describes possible corrective actions.

With knowledge represented as structured text, the knowledge engineer had to determine whether the knowledge contained sufficient detail to be mapped directly into rules or goals, or if further interaction with the pilot was needed to obtain additional details. The structured text representation was considered complete when the knowledge engineer could clearly map the diagnostic knowledge into rules and the remedial knowledge into goals.

The knowledge base was tested in the EPES environmental simulation to verify the completeness and accuracy of the rules and goals. Simulation testing visually confirmed the actions of the ES. The knowledge engineer compared the ES's execution of a procedure with the initial text description of the procedure. When minor discrepancies arose, a tracing facility was used to locate and correct errors in the knowledge base. If a significant problem was detected, the pilot was consulted. The knowledge engineer and the pilot worked together to locate the causes of the problem and to formulate corrections. The pilot provided final verification of the knowledge base.

During knowledge base development for EPES, a number of lessons were learned:

- Although emergency procedures were described separately, often they were interrelated.

- Rules and goals were developed more effectively in parallel and in an iterative cycle.

- The structured text representational format allowed the knowledge engineer to isolate portions of the knowledge base that were incomplete. Specific questions could be generated to obtain the missing knowledge and to minimize interview time.

- Simulation exercises facilitated pilot feedback and aided in capturing knowledge that otherwise would have been difficult to obtain.

Although these processes were used in a flight management domain, this approach to knowledge base development could be applicable to similar domains with dynamic environments (Anderson, et al. 1985).

4.3 NAVIGATION SYSTEMS.

The Knowledge-Based AutoPilot (KBAP) is a NASA/RAE cooperative program established to develop and validate a real-time KBAP. The program investigates implementation and validation issues. The KBAP "provides a simple well defined yet real problem within which to explore, develop, and demonstrate real-time knowledge-based system concepts and validation and verification techniques for mission-critical systems" (Daysh, et al. 1991).

A prototype autopilot was developed using CLIPS, but was found to be too slow for real-time flight control applications. The RAE developed FLEX (a library of FORTRAN 77 subroutines that allow FORTRAN programs to interface with ESs) and has applied it to this application with much improved response time. One of the immediate goals for improving the response time is to use the rules produced under CLIPS and implement them under FLEX. The long-term goal is to apply the conventional software V&V methodology used for flight critical control systems to the KBAP (Daysh, et al. 1991).

4.4 DECISION SUPPORT SYSTEMS.

Curran (1992) states that AI will likely be employed in the future to monitor and assist the pilot. Some of the proposed decision support systems are examined in this section.

4.4.1 Cockpit Assistant System.

The CASSY is a knowledge-based computer aid for flight-planning tasks. The purpose of this system is to support the pilot in complex planning and decision operations during situation assessment and flight replanning operations. This system has been implemented in flight simulators for single pilot and dual pilot testing. Research is being conducted by Dornier and the University of the German Armed Forces in Munich.

Many accidents in highly automated cockpits have been attributed to human error. The intent of CASSY is to use AI to reduce the number of accidents attributed to human error. CASSY functions like an expert copilot to recommend flight plan revisions, to provide warning if the pilot deviates from ATC clearances, and to monitor aircraft systems.

CASSY monitors the aircraft systems to supply the air crew with warnings and advice. It performs situation assessment and planning activities using a knowledge base of ATC rules, airways, instrument approaches, and instrument procedures. CASSY uses voice recognition technology to communicate with the pilot and data link to communicate with ATC. Future plans include providing ATC clearances directly to the pilot and CASSY via data link.

When a pilot deviates from the approved flight plan, CASSY infers whether the deviation was intentional or accidental. The pilot is given verbal advice when a deviation is beyond the prescribed tolerance.

CASSY is comprised of a number of components:

- The dialog manager is the communications interface between the flight crew and CASSY. The initial interface was a speaker-dependent speech recognition system. The goal of this interface is to use a speaker-independent system. CASSY uses synthetic speech to convey messages to the pilot. Different voices are used for different categories of messages. When new routing is recommended, information will be presented to the pilot visually on a map display.

- The automatic flight planner recommends new routes, destinations, and airspeeds. It is activated when ATC instructions are received, system failures occur, or adverse weather forces flight plan changes.

- The piloting expert component determines the actions an expert pilot would take to comply with flight clearances.

- The pilot intent and error recognition component provides a verbal warning to the crew when deviation from the flight plan exceeds a danger boundary.

- The flight status monitor advises the pilot of flight progress as assigned altitudes are reached.

- The systems monitor notifies the crew of malfunctions in aircraft systems.

- The environment monitor will be used in the future to advise pilots of aircraft detected by Traffic Alert and Collision Avoidance Systems (TCASs) and weather conditions from weather radar.

- The execution aid will perform tasks such as adjusting the flight management system and autopilot and setting flaps based on the pilot's verbal commands.

The results of simulator tests have been encouraging. It is the intent of CASSY to simplify the decision-making process, yet keep the pilot in the decision loop. During simulator trials, pilots using CASSY responded to ATC warnings within 5 seconds. Pilots without the system took from 15 to 95 seconds to evaluate the situation and respond.

System tests have been ongoing and future tests will be performed on aircraft. CASSY could be operational within 10 years (Nordwall 1992 and Prevot, Onken, and Dudek 1991).

4.4.2 Diverter System.

In-flight diversions cause considerable pilot workload. After obtaining information from a variety of sources, the pilot must determine present position, and fuel and maintenance status of the aircraft. Also, the pilot may need to consult aircraft handbooks; aircraft performance data; enroute, terminal area, and instrument approach charts; company flight operations; flight service personnel; and air traffic controllers. The new flight plan is expected to make efficient use of manpower, fuel, and time, while fulfilling all applicable constraints. The process is labor intensive and time consuming. Diversion during a critical phase of flight creates additional tasks when workload already is high. If there is inadequate time to obtain all data before initiating the diversion, the pilot may need to base a decision upon incomplete information.

The Diverter System is an AI application that provides pilots with information for making in-flight diversion decisions. In August 1990, a report describing the prototype system was prepared for NASA. The goal of the Diverter System is to use AI and algorithm-based decision aiding to replicate a pilot's information processing and application of logic principles during in-flight diversion planning. The

Diverter System is designed to resemble closely the cognitive and information processing functions used by pilots during diversion functions.

The Diverter System is designed to be "invisible," although continuously active, until a situation arises that requires planning, display, and execution of a diversion. Background activity consists of constant database updates from onboard systems, as well as from ground-based systems through data link communications. Data are not displayed unless the system is activated.

Activation is accomplished either by the pilot or by the Diverter System. A menu option of the flight data Control Display Unit (CDU) allows the pilot to activate the Diverter System. Automatic activation occurs when the reason for diversion is due to a change in onboard aircraft system status or information from data link communications from ground-based sources. Activation annunciation takes place through the aircraft's fault/system status annunciation system.

A functional flow analysis was developed by querying pilots (domain experts) to obtain the methods and logic used during diversion planning. This information was augmented with information from sources such as Federal Aviation Regulations (FARs) and the Airmen's Information Manual (AIM).

A functional analysis was performed to determine how a pilot plans a diversion. It was determined that diversion planning functions include monitoring systems for changes, assessing the impact of changes, assessing response options, and, if required, planning for execution of emergency procedures. After response options are assessed, requirements for course diversion are determined and a plan of action can be generated. Once the diversion is planned, the pilot evaluates the option and possibly executes it.

The Diverter System is designed to evaluate information indicative of the need for diversion, information that is needed to plan the diversion, and changes in system status. If system status evaluation suggests that a diversion is recommended, the system evaluates the nature of the change and its impact on aircraft operations.

Pilot acceptance of the Diverter System is a crucial design issue. The Diverter System is intended to provide a synthetic "associate" crew member to collect and analyze information for diversion planning. The system must be designed to present information in a format that replicates a pilot's planning processes. The Diverter System must be perceived by the flight crew to be an integral part of the flight deck. Information must be represented in a format that is concise, clear, and congruent with formats familiar to pilots. The system must be designed to include sufficient computational power and AI to replicate the cognitive processes of a human crew member. Another design consideration for acceptance of the Diverter System is traceability. When planning discrepancies arise, a pilot should be able to trace the information sources, history, and the rule base application that led to a particular diversion decision.

The Diverter System is designed to use a structured approach to knowledge base evaluation. The knowledge base contains procedural information, which is codified in a set of rules, and declarative knowledge, which is factual information collected from a variety of sources, including human input. Information is evaluated to determine its impact on a possible course of action. If required by the parameters of the problem, the information is used to formulate a decision.

Pilots acquire knowledge through flight training and experience. Each experience reinforces a particular decision strategy. A static rule base was developed representing the information learned by a pilot during flight training. The rule base was augmented with information from flight manuals, the AIM, and FARs. Additions to the Diverter rule base could be made to accommodate the needs or regulations of a specific company.

One important future design consideration is to provide a mechanism for the Diverter System to generate new rules based on "experience." In one sense, Diverter uses dynamic information by collecting information in real time and applying a set of rules to arrive at a recommended course of action. Also, a set of weights is developed as a function of additive components descriptive of the situation.

The Diverter System considers runways, airfields, and routes when making diversion decisions. These areas are evaluated independently, based on factors such as safety, passenger comfort, facilities, schedule, weather, and economy. These factors represent attributes upon which the three areas will be evaluated. The software architecture of the Diverter System is shown in figure 4.4-1.

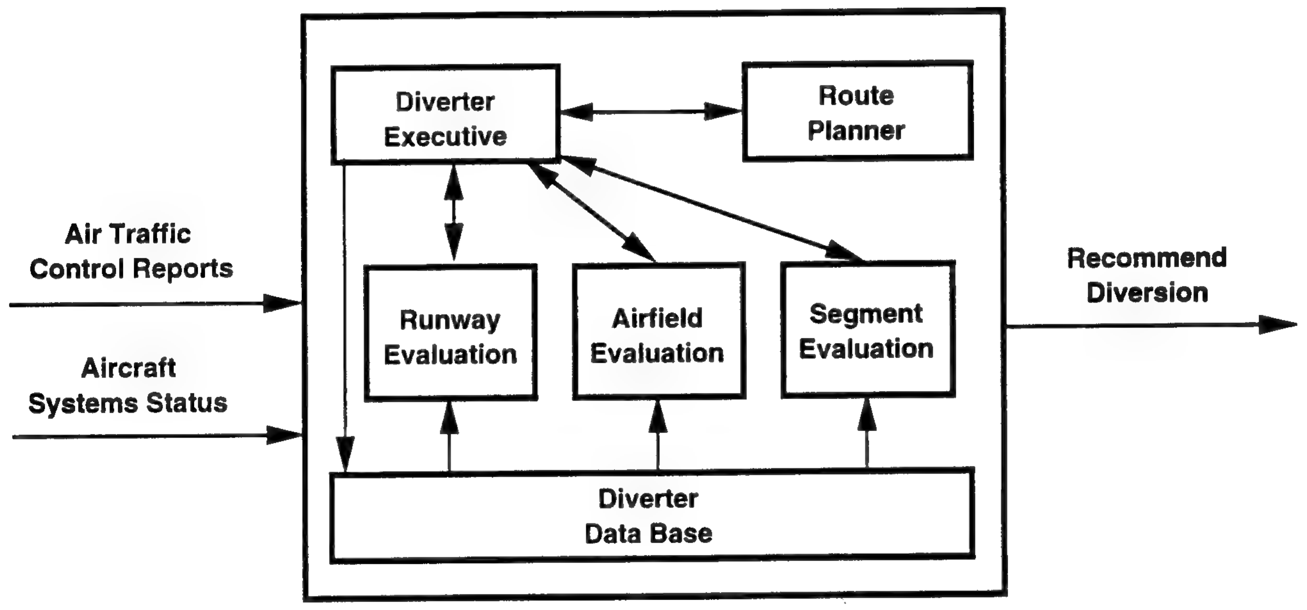


FIGURE 4.4-1. DIVERTER SOFTWARE ARCHITECTURE
(Rudolph, Homoki, and Sexton 1990)

The areas of runways and airfields are broken down into a set of attributes for each area. Weights are assigned to attributes to indicate their relative importance. The weights represent a method of controlling the Diverter System's behavior. During the planning process, the runways and airfields are ordered based on their values for the appropriate attributes. Then they are assigned a rank for each attribute. The rank is multiplied by the weight, and the result is the score for that runway or airfield for that attribute.

Route evaluation is handled differently from runway and airfield evaluation. Routes are composed of segments. Each segment is assigned a value, which represents the cost of the segment to the planner. Cost is based on the attributes of a segment. Each segment attribute is assigned a weight. If a segment attribute is true, the weight of the attribute is multiplied by the segment distance and added to the segment cost. When all of the segments have costs, a route planning algorithm is used to find the least costly route to the airfield under consideration.

The Diverter prototype is capable of combining available knowledge and using built-in knowledge to determine the best combination of runway, airfield, and route. The resultant route is then suggested to the pilot. In an actual system, runway, airfield, and aircraft attributes would be stored onboard, while information, such as weather and navigation aid status, would be obtained through data link or pilot input. The current Diverter System prototype did not attempt to define how this information transfer would take place. Also, the form in which information would be presented was not defined.

Following are areas of future development for the Diverter System:

- Using direct routing
- Allowing the pilot to use pieces of the Diverter System as a tool
- Modifying software to include more complete flight plans
- Performing additional research on assigning weights to attributes
- Developing an intelligent system that can learn from itself

The Diverter System prototype represents an AI application that can provide pilots with decision-making help in a complex environment. In addition, the Diverter System offers the pilot an information resource and a tool for enhanced planning capabilities (Rudolph, Homoki, and Sexton 1990).

4.4.3 Flight-Plan Interactive Negotiation and Decision-Aiding System.

The Flight-Plan Interactive Negotiation and Decision-Aiding System for Enroute Rerouting (FINDER) is being developed by Sextant Avionics and will be tested by Air France pilots when completed. FINDER is an interactive system that uses ES techniques, as well as conventional code written in C and C++.

FINDER's purpose is to suggest a limited number of satisfactory solutions for crew members facing a diversion situation (FINDER 1) or, more generally, an enroute replanning situation (FINDER 2). The system focuses on managing existing data and making decisions that aid the pilot. Following is information that FINDER will use to provide pilot assistance (Bittermann, et al. 1994):

- Fuel capacity
- Weather forecasts
- Ground facilities
- Regulations
- Passenger-based constraints

The diversion problem is clearly stated here (Bittermann, et al. 1992):

> ...when pilots are confronted with diversion situations, they must find a solution which theoretically integrates all the parameters within a limited time and is based solely on their personal experience. Their attention is mainly concentrated on the critical considerations that may affect the aircraft's survival (fuel, weather).

The system will also, upon request, give reasons for a particular selection and allow modification by pilot/system dialog.

It is assumed that FINDER will have data link connections available with the airline owner, ATC, and possible destination airports. In addition, information is necessary from the Central Maintenance System and the Electronic Library System. FINDER will also investigate the impact of the evolving negotiation process between the crew and ATC.

A diversion is generated by FINDER using a two-phase decision process. Figure 4.4-2 shows how the process works.

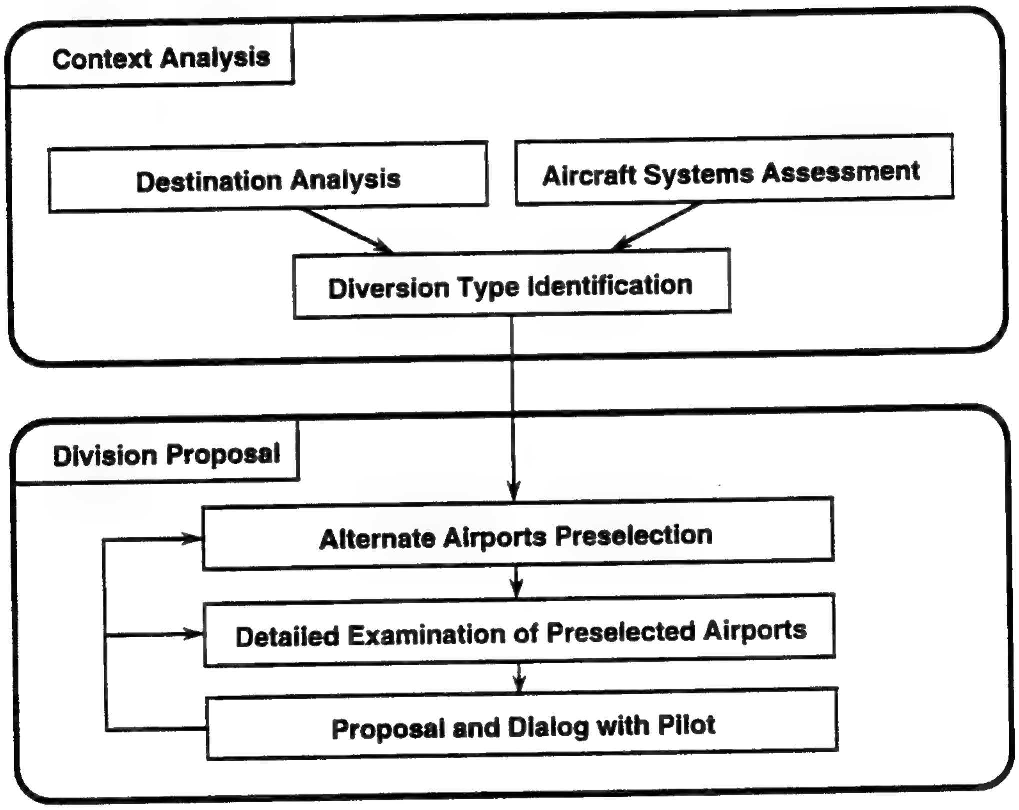

Context Analysis
Destination Analysis
Aircraft Systems Assessment
Diversion Type Identification
Division Proposal
Alternate Airports Preselection
Detailed Examination of Preselected Airports
Proposal and Dialog with Pilot

FIGURE 4.4-2. DIVERSION PROPOSAL GENERATION
(Bittermann, et al. 1992)

The first phase, Context Analysis, examines the environment as completely as possible. This includes the condition of the aircraft and facts about the possible destination airports. Following are factors taken into account by the Context Analysis phase (Bittermann, et al. 1992):

Destination Analysis

- Weather conditions assessment
- Infrastructures assessment
- Emergency facilities assessment

Aircraft Systems Assessment

- Identification of technical problems
- Estimation of aircraft range
- Assessment of passenger constraints
- Assessment of aircraft load constraints
- Assessment of crew constraints

The type of action to propose is decided based on these information lists.

The second phase, Diversion Proposal, starts with the type of action determined by the Context Analysis phase and seeks to arrive at a satisfactory solution in light of all known facts. The first stage of the Diversion Proposal selects a number of possible destination airports, based on the destination analysis information for those airports. Using optimization heuristics, the optimum route is selected for the possible destination airports.

The last stage of this phase is the destination airport proposal and dialog with the pilot. A Graphical User Interface (GUI) is used for the pilot-system interface. A list of proposed airports is displayed by the GUI. If the pilot selects one of the proposed airports, the system will handle the necessary interface with ATC via the data link.

If the pilot does not select one of the airports, he or she can modify the parameters used in decision making. This will cause the system to regenerate the list of proposed airports in light of these new parameters. This new list will then be displayed for selection or further modification by the pilot.

The concept of diversion assistance during the FINDER 1 project led to the FINDER 2 project. FINDER 2 is devoted to developing an enhanced real-time demonstrator with extended replanning features (avoidance and optimization) and an enriched simulation environment (weather events, predictions and reports, regulated areas, and ATC and operation constraints).

The situation assessment module has been extended to overall environment and aircraft monitoring. It identifies potential problems, proposes corrective actions, and provides the crew with a comprehensive picture of the context evolution.

Each solution generated is maintained in association with the context and aircraft evolution (i.e., if an unaccessible airport becomes available while in diversion mode, the previously proposed solutions are automatically updated).

The pilot-system interface is hosted by an enhanced Electronic Library System Pilot Access Terminal. The logic of the dialog has been defined according to cognitive modelling procedures. The software architecture takes advantage of concurrency between the modules of the system's kernel (mainly situation assessment, replanning, and dialog). Human-like explanations are continually provided by the system on the pilot's request.

4.4.4 Pilot's Associate Program.

A significant amount of aviation-related AI research in recent years has originated from programs funded by the U.S. Air Force (USAF) Aeronautical Systems Division. One such program, Pave Pace, will investigate and validate many areas of new technology, including AI algorithms, to assist in mission decision making and NN parallel-processing capability for adaptive learning systems (Curran 1992).

The USAF and the ARPA awarded a contract to a team from Lockheed and a team from McDonnell Douglas Corporation. This program, the Pilot's Associate (PA), has the goals of defining, designing, and demonstrating the application of machine intelligence to the assistance of advanced fighter aircraft pilots (Leavitt and Smith 1989). The Lockheed team was awarded a contract to develop an ES that tailored its response to the pilot's particular style of flying.

This particular system had technically difficult constraints. It had to react to inputs and communicate with the pilot rapidly. In addition, the knowledge base was designed to receive inputs from a variety of systems. The construction of the knowledge base relied on inputs from human factors experts, psychologists, engineers, and pilots.

The following six software modules were required to make the design manageable (Curran 1992):

- Tactics Planner
- Mission Planner
- System Status Module
- Situation Assessor
- Mission Manager
- Pilot Vehicle Interface (PVI)

The tactics planner provides information to the pilot about maneuvers such as attacking or evading. Recommended flight paths are created by the mission planner. A system status module monitors engines for various malfunctions, such as excess vibration and overheating. The situation assessor evaluates targets and threats. A mission manager is used to coordinate communication among the other software modules.

The PVI was developed to interface the PA to the end user, the pilot. This module interprets the intent of the pilot and determines the amount of information to present to the pilot at a given instant in time. If a pilot deviates from the planned mission, the PVI modules will flag this condition and configure the displays with the relevant information. If the PVI module detects a malfunction during a period of high workload, it decides whether or not to inform the pilot immediately. In addition, knowing the intent of the pilot, the PVI module will perform the following functions:

- Detect flying errors
- Determine the seriousness of the error
- Report the error to the pilot
- Correct the error if authorized by the pilot

A key element in the PVI module architecture is the Plan-Goal Graph (PGG), which links the pilot's actions with a particular goal. There are 300 nodes in the PGG consisting of action-intent relationships.

A goal is implied by a particular action. Links are established among the graph nodes by 800 IF THEN knowledge base rules. Each input, or action, is used to clarify the pilot's intent and reach a particular goal. Figure 4.4-3 is a pilot intent illustration for a goal of having communication.

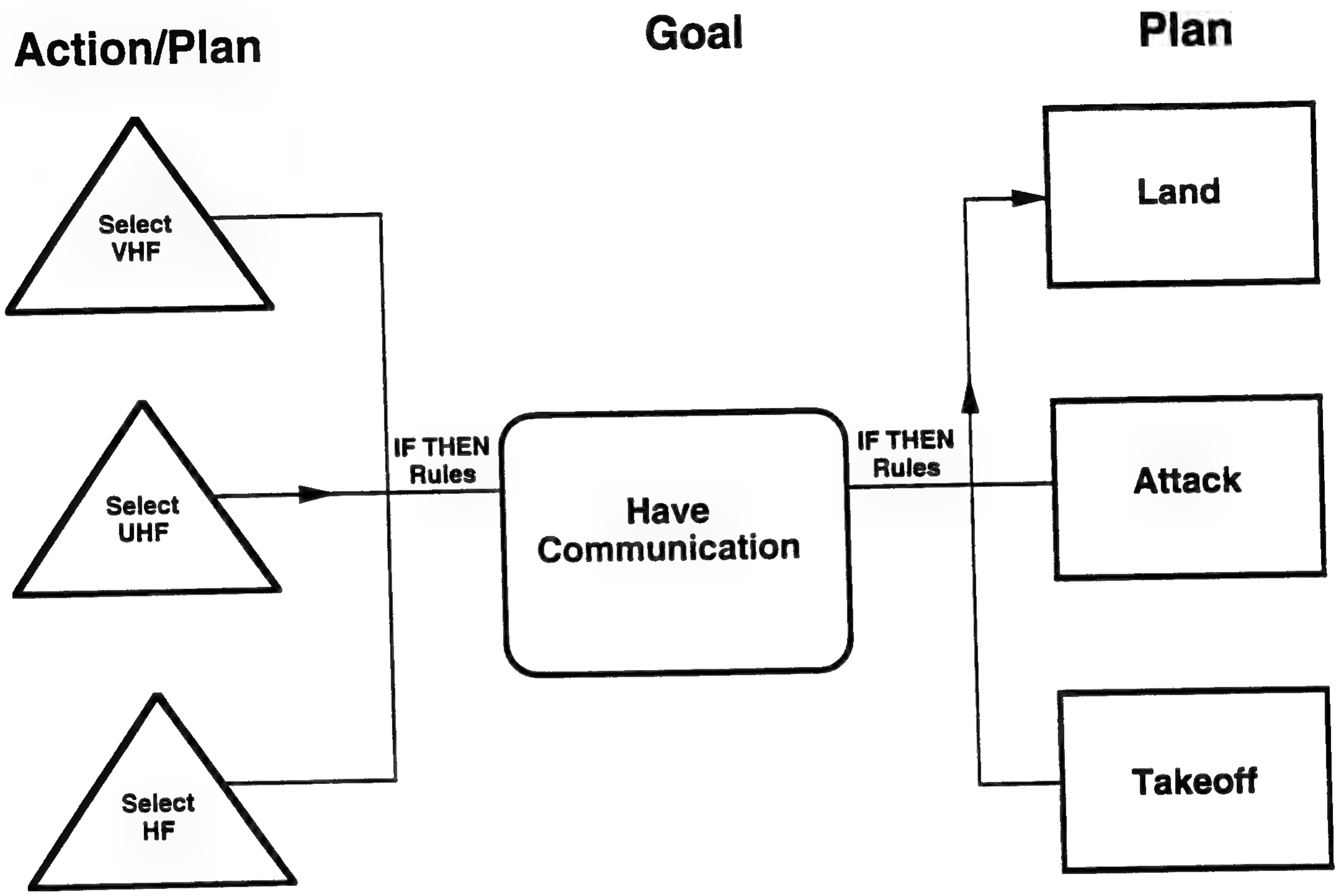


FIGURE 4.4-3. PILOT INTENT DIAGRAM
(Rouse, Geddes, and Hammer 1990)

When a pilot action can not be explained by the PVI module, an error monitor is invoked. If the error monitor decides that an error does indeed exist, a determination is made about what error-related information should be given to the pilot.

Other characteristics are modeled by the system, including resource and performance. The resource model "tries to predict whether a pilot could devote attention-vision, hearing, and hand movement to a particular task." The performance model tries to determine the amount of time the pilot would take to perform the recommended task (Rouse, Geddes, and Hammer 1990).

One goal of the PA program is to have all of the software running on an avionic processor using a real-time procedural language, such as C or Ada (Rouse, Geddes, and Hammer 1990).

4.4.5 Rotorcraft Pilot's Associate Program.

The Rotorcraft Pilot's Associate (RPA), a program sponsored by the U.S. Army, has similar goals to the PA program. The Army requested proposals from industry to develop a system that would alleviate the workload of rotorcraft combat pilots. The RPA proposes the use of a knowledge-based system to draw upon the knowledge of experts with prior experience in this discipline.

During the Vietnam era, a number of lessons were learned about combat avionics. It was found that when fighter aircraft and rotorcraft approached a combat zone, pilots were shutting down some systems. The systems remaining "on" were those that worked automatically and reliably, yet kept the pilot fully aware of system status. These systems were advisory in nature and the pilot always was in command of the system.

The RPA program follows this design trend. The system monitors the rotorcraft and provides situation awareness. If an unknown target is encountered, the system suggests the correct response. The pilot can choose to accept the response or reject it. If the response is accepted, the system implements it automatically. The purpose of this system is to reduce cockpit workload and free the pilot to concentrate on tasks necessary for combat survival. Figure 4.4-4 identifies the basic functions included in the RPA.

The knowledge base will require detailed study and definition of the decisions that a pilot must make in combat aviation. In addition, difficult issues, such as deciding which functions to automate and which are best left to the pilot, will need to be faced. The growth of AI technology makes the RPA project viable for future combat rotorcraft (Harvey 1993).

4.5 INTELLIGENT MONITORING AND DIAGNOSTIC SYSTEMS.

As aircraft systems become more complex, difficulties have increased for system maintenance tasks. False alarms consume as much as 50 percent of maintenance resources, while troubleshooting actions take as much as 50 percent of total labor hours spent for repair. Accurate and efficient diagnostic systems are needed.

Early attempts to develop computer-assisted diagnostic systems were based on the reasoning processes of experts. Heuristics were coded into rule-based ESs. MYCIN and INTERNIST are examples of diagnostic rule-based systems from the medical field. The Naval Research Laboratory recently has produced the Fault Isolation System, which reasons from a set of causal rules rather than traditional heuristics.

Developing rule bases for complex system diagnostics is difficult. Few experts possess ample understanding of new systems to provide adequate diagnostic rules. If multiple experts are used, it is difficult to gain consensus about the rules. These problems may be addressed by using rules based on causal relationships between failures and test information and by developing rules from examples of actual or simulated failures. However, providing sufficient examples to produce adequate fault trees may be impractical.

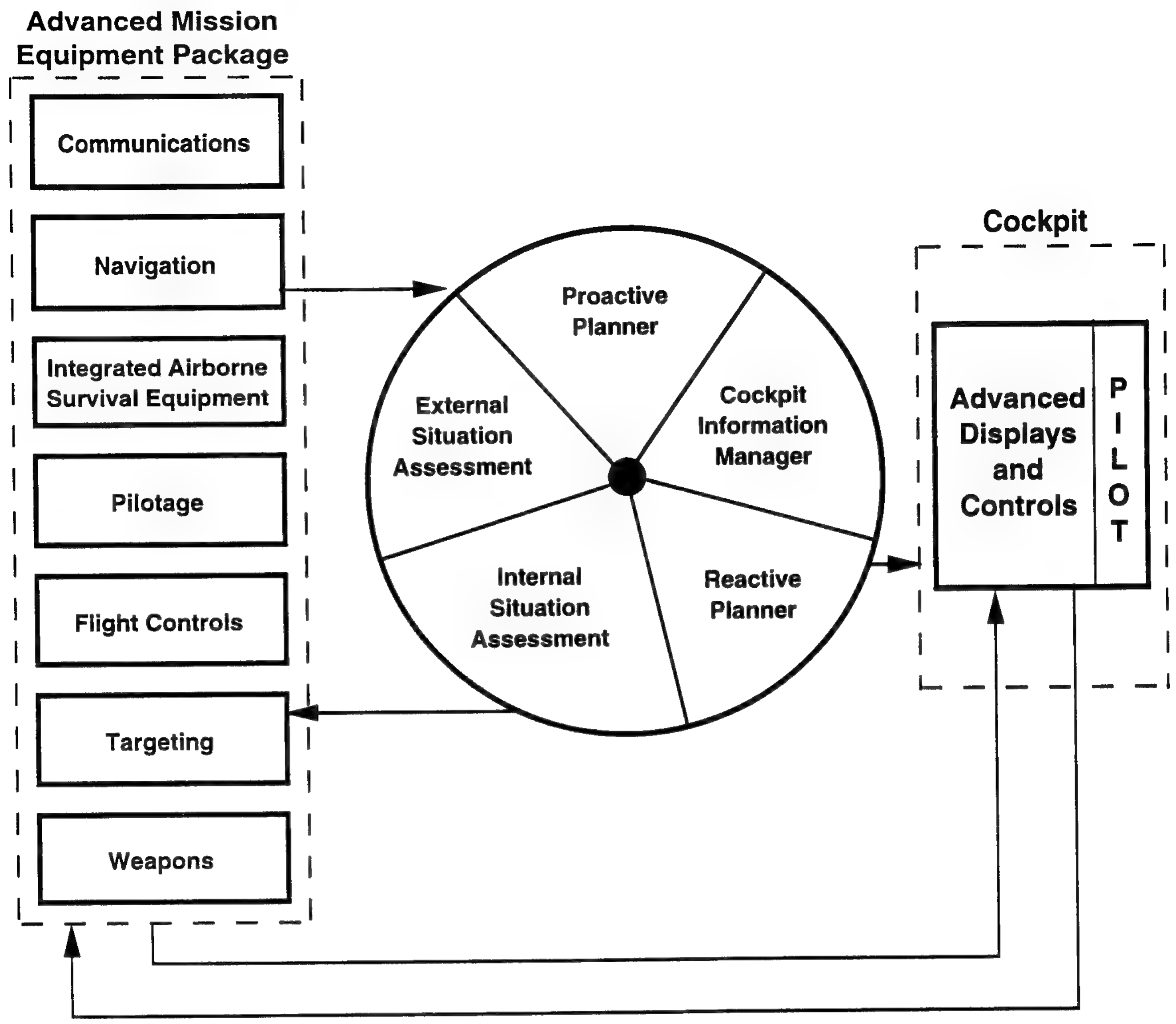

Advanced Mission Equipment Package
Communications
Navigation
Integrated Airborne Survival Equipment
Pilotage
Flight Controls
Targeting
Weapons
Proactive Planner
Cockpit Information Manager
Reactive Planner
Internal Situation Assessment
External Situation Assessment
Cockpit
Advanced Displays and Controls
PILOT

FIGURE 4.4-4. RPA DEMONSTRATOR CONCEPT
(Harvey 1993)

Model-based diagnostic aids are an alternative to rule-based systems. Model-based systems use mathematical representations of the system to be diagnosed. Generally, models are structural or behavioral. Structural models describe the connectivity of a system to be tested. When a component fails in a structural model, all tests fed by that component will detect the failure. Behavioral models describe system performance in terms of mathematical functions. The circuit is diagnosed by applying several different inputs to the system, examining outputs at several points, and identifying discrepancies between actual and expected behavior.

A combined model uses features of the structural model and the behavioral model. System connectivity is similar to the structural model, but tests are not limited to pass or fail outputs. Components of the model are described in terms of their transfer functions. These transfer functions define a set of constraints to be applied in performing a diagnosis. Failure candidates are identified (via an algorithm) as components that violate the constraints of the behavioral model or that may have led to the violation. Diagnosis proceeds from the system outputs back to the inputs until the failure is identified.

Another diagnostic model is the information-flow model. The information-flow model includes a structural and a logical representation of the system being analyzed. Groups of logically related tests and conclusions and logical values for test and fault isolation conclusions are defined by the model. Knowledge acquisition bottlenecks are minimized by incorporating elements of machine learning for model production and correction. The model applies an inference engine to the information-flow model so that fault isolation sequences can be optimized and conclusions drawn from tests performed. The information-flow model has been applied successfully to avionic systems, such as the Combined Altitude Radar Altimeter, the stability augmentation system of the UH-60A Black Hawk helicopter, the APG-63 radar set for the F-15, and several podded electronic warfare systems. In addition, this model was applied to the entire B-2 bomber avionic system.

ESs are being developed for a number of aspects of avionics diagnostic testing. ESs monitor the results of Built-In Test (BIT) applications to provide better localization capability. Performance monitoring and fault locating software determine the cause of anomalies as they occur. In the future, software will be integrated into onboard maintenance systems to allow online diagnosis, reconfiguration, and repair.

Another application of ESs is production of Interactive Electronic Technical Manuals (IETMs) and Portable Maintenance Aids (PMAs). IETMs and PMAs combine online technical information with maintenance facilities.

ESs are beginning to be used in avionics testing as test executives for intelligent Automatic Test Equipment (ATE). The ATE systems' capabilities are as follows:

- Learn from experience
- Devise and execute test strategies dynamically
- Correct errors
- Explain and validate test choices

With increasing use of ESs in ATE, IEEE is developing a standard for AI-based ATE, which addresses issues such as the following (Sheppard and Simpson 1992):

- Data and knowledge management services
- Data and knowledge interchange services
- Human services
- Program services
- Network services
- Model base specifications

4.5.1 Rotorcraft Transmission Health Monitoring System.

As commercial and military aircraft age, there is increased emphasis on maintenance and safety issues. One important problem in this area is predicting impending faults in helicopter transmissions. In new aircraft, advanced diagnostics in which continuously monitored volumes of data must be reduced to provide meaningful information to pilots or ground crews also are important.

The Office of Naval Research (ONR) has initiated an aircraft health monitoring program that is investigating the use of NNs in mechanical diagnostic systems. Reduced complexity, speed, and cost were cited as advantages of using NNs in diagnostic systems. Reduced complexity would increase the reliability of a diagnostic system. NNs will allow real-time diagnostics to be performed onboard the aircraft. These diagnostics will fuse data from the power train, drive train, structure, rotor system, and oil systems. As ICs to support this technology become mass produced, the cost of this technology will be increasingly affordable.

In June 1992, ONR contractors began work on an NN application to detect helicopter gearbox faults. Phase 1 has concentrated on detection and feature extraction. Subsequent phases of the ONR program will focus on classification, network design, and potential migration to a commercial product. Researchers in phase 1 of this project have suggested use of a three-level, feed-forward model and similar extraction techniques.

A hierarchical NN approach using three-layer perceptrons was investigated. A perceptron is a pattern recognition mechanism. It is capable of learning by means of a feedback system that reinforces correct answers and discourages incorrect answers. Four feature extractors were coupled with their own NNs in a first layer. The outputs from the first layer were merged into a second layer net for fusion processing. Using the NN classifier, enough data was generated in 20 rotations of the shaft to differentiate the transmission states.

In another approach, time domain data from three sensor channels was run through a 256-point Fast Fourier Transform (FFT), generating 128 spectral components. These components formed inputs to the network.

In a third approach, an NN was trained on a sample set of transmission data using a commercial software package, CLASS, from Barron Associates, Incorporated. This software uses polynomial nodes and a constrained multiclass, minimum logistic loss criterion. This is then coupled with an output-layer transformation to compute class probabilities for each output category. These probabilities always sum to unity. Signal preprocessing and feature extraction were kept to a minimum for this preliminary investigation. Principal component analysis was used to determine a subset of these features that accounted for most of the variation in the data. With approximately 20 features, perfect classification accuracy was obtained on both training and evaluation data.

Continuous Wavelet Transformation (CWT) is a technology that selects features for an NN classifier. In this particular application, the CWT reduced the probability of false alarms and misdetections to zero.

Preliminary ONR reports indicate a need for more and higher quality data with both fault and normal conditions. In the future, implementation of a data fusion approach for health monitoring will involve ESs, blackboards, NNs, and other AI technologies. Research still is needed in areas such as smart sensors, improved system-state recording, advanced algorithms, and faster data sampling and processing. Programs such as the ONR research in transmission diagnosis can enhance safety by using AI to improve health and usage monitoring (Rock, Malkoff, and Stewart 1993).

4.5.2 Technical Expert Aircraft Maintenance System.

The complexity of modern combat aircraft avionic systems makes diagnosis and repair of these systems a difficult process. Knowledge of the aircraft, as well as a broad base of repair experience, is needed. Faults that occur during initial testing of newly manufactured aircraft are especially difficult to isolate, since the assumption that a failed system once worked correctly cannot be made. Typically, after each test flight, a team of mechanics fixes any problems that were discovered. The team of mechanics is supported by a staff of engineers who are specialists for each specific system.

The TEAMS, developed by McDonnell Aircraft Company, is an interactive ES that supports problem diagnosis in new aircraft. Using an ES in fault diagnosis supports the mechanic's diagnostic decision-making process and reduces the need for engineering support.

Development of ES technology applied to maintenance systems occurred in several stages. The first stage involved an evaluation of the technology and its expected impact on the aircraft delivery process. An ES was developed for the Communication, Navigation, and Identification (CNI) suite of systems. During a 3-month study period designed to assess the impact of the ES, it was determined that the CNI ES could result in a significant reduction in diagnostic time. The study also showed a reduction in the number of test flights required to sell an aircraft and a reduction in the number of spares required to be kept in inventory.

During stage 2, additional ESs were developed based on the results of stage 1. Flight controls and radar were selected for stage 2 because they produce the greatest number of flight faults in the avionic systems and isolation of failures in these systems requires considerable engineering support. This ES was developed using CLIPS. Stage 3 involved expanding the ES concept to other programs.

The TEAMS ES development encompassed four phases: project identification, knowledge acquisition, knowledge codification, and testing (see figure 4.5-1). Project identification involved determining the aircraft system that could benefit most from a fault isolation ES. To accomplish this task, flight ramp aircraft fault history was reviewed to identify systems with high failure rates. Also, flight ramp mechanics and engineers were interviewed to identify systems that were difficult to troubleshoot and repair. Another activity during project identification was to identify suitable experts. A knowledge engineer was assigned to the system to work closely with the expert.

Phase 2 was knowledge acquisition. Each ES developed in this project was based on a different type of knowledge. This resulted in diagnostic approaches that were unique to the characteristics of each aircraft system. For example, the CNI ES was a fault-tree-based knowledge base. The most likely failure cause was identified by examining symptoms and the test results. The flight control's ES was based heavily on the results of BITs. The radar ES was a hybrid system that used fault-tree type knowledge and BIT code-based knowledge.

Phase 3 entailed knowledge codification. Knowledge acquired from the expert was coded into CLIPS knowledge bases by the knowledge engineer. When knowledge codification was complete, the expert would review the system and recommend changes in the knowledge. The outcome of this process usually was additional knowledge acquisition.

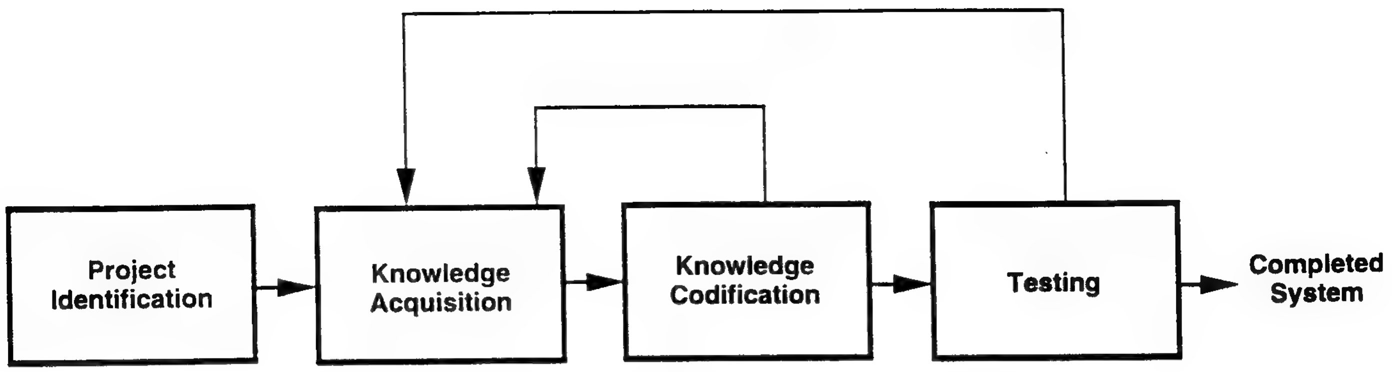


FIGURE 4.5-1. SYSTEM DEVELOPMENT ACTIVITIES
(Lischke and Meyer 1992)

Testing was the final phase in ES development. Alpha testing primarily was concerned with the accuracy of the knowledge. Alpha testing was performed by the expert, the knowledge engineer, and others experienced in the domain area. Beta testing, performed by end users, focuses on usability of the ES. Human factors, such as readability and understandability of screens, were examined. The purpose of this testing was to ensure that the system would be usable by non-experts in the domain area.

A number of approaches were used to involve the end user in the project and to encourage user acceptance of TEAMS. A training course was developed as soon as the CNI system was complete. The course focused on basic operation of the system and using the system in a simulated environment to find flight faults. A comment feature was added to TEAMS. This feature allowed users to enter a remark at any time during their use of the ES. The comment feature provided a valuable source of feedback. When comments were made, the designers were contacted for clarification (Lischke and Meyer 1992).

5. CERTIFICATION CONSIDERATIONS FOR ARTIFICIAL INTELLIGENCE.

Depending on the intended use of a software-based system, incorrect software can result in significant safety or economic impacts. When safety issues are a primary consideration, a software-based system may be required to undergo certification corresponding to its application and criticality. Critical applications of software require rigorous attention to methods for reducing the probability of failure and ensuring the safety of those potentially affected by the failure of such software. This is no less true when applied to AI-based software. A "bug" in the AI-based software in a research lab may have minimal effects. However, a bug in the ES used to control stock market buying and selling or in the ES used to diagnose and monitor nuclear power generation systems may have far-reaching consequences.

For airborne software, certification is the process of obtaining FAA approval for the design, manufacture, and/or sale of a part, subsystem, system, or aircraft by establishing that it complies with applicable government regulations. The purpose of certification is to demonstrate and record that the total aircraft is suitable and safe for use (Elwell, et al. 1991). Key to certification is proving the software to be reliable and high in quality by a thorough examination of the methods used in designing, coding, testing, and life cycle planning.

Much research has taken place related to verification, validation, and testing of AI-based ESs. This section presents some of the methods and considerations relating to this discipline for AI-based software. A background of V&V techniques for conventional software is provided. Additionally, this section highlights some of the concerns relating to the proposed use of AI technology in the cockpit.

5.1 CERTIFICATION AND FEDERAL AVIATION REGULATIONS.

Proposed airborne applications of AI-based technology would be considered avionic systems. The requirements for certification of avionic systems are covered in the FARs. Following are some of the relevant FARs:

- Part 21, "Certification Procedures for Products and Parts"
- Part 23, "Airworthiness Standards: Normal, Utility, and Acrobatic Category Airplanes"
- Part 25, "Airworthiness Standards: Transport Category Airplanes"
- Part 27, "Airworthiness Standards: Normal Category Rotorcraft"
- Part 29, "Airworthiness Standards: Transport Category Rotorcraft"
- Part 91, "General Operating and Flight Rules"
- Part 121, "Certification and Operation: Domestic, Flag, and Supplemental Air Carriers and Commercial Operators of Large Aircraft"
- Part 125, "Certification and Operation: Airplanes Having a Seating Capacity of 20 or More Passengers or a Maximum Payload Capacity of 6,000 Pounds or More"

- Part 127, "Certification and Operations of Scheduled Air Carriers with Helicopters"
- Part 135, "Air Taxi Operators and Commercial Operators of Small Aircraft"

When a major design effort is required to develop a system, the integrity of the aircraft into which it will be installed is in question. Thus, one of two type certification processes must be followed to receive a certificate. For totally new designs, or changes that are so extensive as to require a complete reinvestigation of the design, the developer must follow the process required to obtain a Type Certificate (TC) for the aircraft. For major changes (as defined in FAR Part 21, section 93) to a system previously approved under a TC, the developer can follow a simpler process to obtain a Supplemental Type Certificate (STC).

The Technical Standing Order (TSO) authorization method of approving components was developed to allow manufacturers to substitute equivalent components "off-the-shelf" without jeopardizing the existing TC. Since a TSO authorization request must be processed within 30 days and does not require integration testing, manufacturers use this method rather than Type Certification whenever possible.

In the days of simpler aircraft design, TSO authorizations were adequate. Now, however, digital avionic systems are more complex and require elaborate testing procedures. To improve the TSO approval process, Aircraft Certification Offices (ACOs) are becoming more involved in approving new digital systems. They are reviewing V&V plans for TSO packages and are working more closely with the manufacturers. The ACOs are suggesting that system integration test plans be required for substitutions in integrated digital systems.

As avionic systems become more complex, the manufacturer must ensure that the system will function as intended within its operating environment. Manufacturers' validation facilities will play a major role in establishing the requirements for integration testing, since the functions of digital avionic systems will be simulated there. The certification requirements will expand for such systems to reflect the FAA's concern that the systems safely perform their functions once they are installed in an aircraft.

The system to be certificated can be simple or complex. The FARs stipulate which process must be followed in each case. Although the manufacturer may refer to the FARs to decide which approval should be sought, often a Certification Engineer (CE) recommends which application the manufacturer should submit. The authority for determining whether a change constitutes a modification or a redesign, and whether a redesign is minor or major, rests with the ACO.

By way of example, the Boeing 777 is being developed under a TC program. Commercial aircraft operating under FAR Part 121 rules are being retrofitted with TCASs. This system was developed under STC programs (Elwell, et al. 1991).

It is unclear whether a pilot assistance system presented for certification would require an STC or a TSO. An STC could be appropriate since it may be viewed as a major change to a previously approved system. To be considered under a TSO a similar system must exist. No similar system exists, but it may be possible to integrate an ES into an existing aircraft Line Replaceable Unit (LRU) and use an existing CDU for system I/O.

5.2 CERTIFICATION GUIDELINES.

In general, to show compliance with the FARs, a system or component is subjected to environmental tests, software tests, failure analyses, and other testing, as deemed necessary. The FAA relies on the manufacturer to conduct testing. If the FAA approves new tests, the manufacturer must comply with them. These tests are identified in guidelines adopted by the FAA. Organizations such as Aeronautical Radio, Incorporated (ARINC), Engineering Society for Advancing Mobility Land Sea Air and Space (SAE), and Requirements and Technical Concepts for Aviation (RTCA) produce such guidelines. Some guidelines which may be applied to AI-based systems are identified below. Further details of the certification process can be found in DOT/FAA/CT-91/19, "Avionic Data Bus Integration Technology."

5.2.1 Hardware Guidelines.

In the past, to certificate an airplane, inspectors and engineers had to understand avionics based on analog electronic systems driving mechanical, pneumatic, and/or hydraulic systems. One of the currently used guidelines for the certification of avionic hardware is RTCA/DO-160C, "Environmental Conditions and Test Procedures for Airborne Equipment." RTCA/DO-160C addresses the effects of magnetic fields, voltage spikes, and induced voltages on electronic components. Any airborne hardware is subject to its requirements. There are also a number of other issues relating to hardware that influence the safety of flight.

Another guideline is the SAE's Aerospace Recommended Practice (ARP) 1834. It defines Fault and Failure Analysis (F/FA) techniques for digital hardware. Since digital systems are fault prone, the FAA has decided that fault analysis should be employed during the certification process. FAR Parts 23, 25, 27, and 29, section 1309, and Advisory Circular (AC) 25.1309-1A express the need for fault analysis and ARP1834 has provided a means for conducting such an analysis.

Today's digital systems are more complex than those for which the tests were designed. Digital hardware complexity has increased to the point that traditional design methods can no longer be used. Automated tools are required for design, layout, and testing of complex ICs and circuit boards. According to Keller (1992):

> Some electronics systems themselves have attained a complexity level such that traditional lab and flight testing methods cannot realistically be extensive enough to show compliance to existing requirements.

New guidelines, however, may be adopted in the future as a result of RTCA's Special Committee (SC) 180, which is formulating design assurance guidance for complex electronic hardware used in airborne systems (RTCA Paper No. 548-93/SC180-18 1994). Following are some of the topics to be addressed in this effort:

- Hardware development life cycle
- Planning process
- Development process
- Verification process
- Configuration Management (CM) process

- QA process
- Certification liaison process

5.2.2 Software Guidelines.

This same explosive growth that occurred with digital hardware technology has occurred in the software domain as well, to the extent that "the probability of a hazardous error in software cannot be quantified" (Keller 1992).

Keller identifies three major issues relating to certification of software-based systems:

- Method of compliance demonstration
- Determination of the rigor required to ensure software integrity
- Determination of the software contribution to systems hazards

A method of compliance demonstration commonly used by manufacturers to meet FAA requirements is found in "Software Considerations in Airborne Systems and Equipment Certification" (RTCA/DO-178B 1992). This document was prepared by SC 167 of RTCA and was adopted as an official guideline in FAA AC 20-115C. Since software is generally too complex to test for errors, guidance is given in this document on acceptable processes used for developing airborne software. The software developer is required to be in compliance with this guideline from the initial stages of development and must have the appropriate artifacts to demonstrate the amount of development rigor (Keller 1992).

Determination of the rigor required to ensure software integrity is done in concert with the participating ACO. RTCA/DO-178B defines five categories of failure conditions, ranging from "Catastrophic" to "No Effect." Associated with each of these categories is a corresponding level, ranging from Level A through Level E. Level A, the most critical level, is defined as:

> Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a catastrophic failure condition for the aircraft.

Software that is determined to have no effect on the crew workload or the operational capability of the aircraft is defined as Level E. Once the software level determination is made, software at Level E is not subject to any additional guidelines of RTCA/DO-178B. The remaining three levels, from Level D through Level B, define software having an increasingly greater effect on flight safety.

RTCA/DO-178B addresses guidelines for producing airborne software that will meet airworthiness requirements. It addresses software life cycle processes, the activities associated with those process objectives, and the evidence that shows that the objectives have been met. It is a document that addresses all the critical stages of software development. The document also includes tables indicating the various processes, their objectives, and their outputs based on software level for the following categories:

- Software planning process
- Software development process
- Verification of outputs for software requirements, design, and coding process

- Testing of outputs of integration process
- Verification of verification process results
- Software CM process
- Software QA process
- Certification liaison process

5.2.3 Software Guidelines and Artificial Intelligence.

5.2.3.1 RTCA/DO-178B.

Existing certification guidelines were not designed with AI or knowledge-based software in view. The unique characteristics associated with the development and testing of AI-based software are not covered in RTCA/DO-178B, although AI has been identified as a technology that may receive attention in a future revision (DeWalt 1992). Manufacturers wishing to implement systems incorporating AI are, therefore, faced with certain difficulties when considering the certification process.

As an example, RTCA/DO-178B requires that algorithms be checked for accuracy if they are classified at level A, B, or C. While ESs may contain algorithms, they also contain knowledge bases which should be subjected to the same processes as conventional software. Just as algorithms are checked for accuracy, knowledge base accuracy also needs to be verified. For software criticality levels A or B, this verification should be satisfied by independent verification. This may be difficult for ESs, since domain experts are scarce and their knowledge can be contradictory. In addition, eliciting, capturing, and representing knowledge from domain experts can be a difficult task. Not even the expert can know if all the relevant information has been embodied in the knowledge base. Acceptable methods for knowledge base verification need to be addressed.

While AI is not specifically addressed, the major processes that are applied to conventional software are also applied to AI-based software. Additionally, progress in development tool capability requires that advanced development tools and methods be addressed. ESs are developed with tools that allow system definition at a very high level, without the manual generation of traditional source code. These tools are considered by RTCA/DO-178B, where it states:

> ...if Source Code is generated directly from high-level requirements, then the high-level requirements are also considered low-level requirements, and the guidelines for low-level requirements also apply.

This ensures that none of the requirements for software development are overlooked due to the tools used for development.

ESs are similar to advanced development tools, since there are few intermediate levels when ESs are viewed from the requirements level through the executable code level. While the number of products generated during the development of ESs are few as compared with conventional software, the trend in conventional software is to design systems at higher levels and reduce the number of steps in the development process by using automated design tools. Regardless of the number of development steps, tool qualification issues also need to be addressed, whether for conventional or ES-based systems. These and other related topics should receive coverage in a future revision of RTCA/DO-178.

5.2.3.2 DOD-STD-2167A.

DOD-STD-2167A establishes uniform requirements for software development over a system's life cycle. ES developers can face problems when compliance with DOD-STD-2167A is required.

A review of the structure of conventional software development can identify products that are required to comply with the DOD standard. Five categories of the development cycle for knowledge-based systems include the following (Carlton 1987):

- Identification – Identifies the problem for which a solution is required. It helps to focus the knowledge acquisition process
- Conceptualization – Identifies the relationships needed to describe the domain's problem solving process.
- Formalization – Involves translating the relationships identified in the conceptualization phase into a knowledge structure.
- Implementation – Deals with organizing the knowledge and developing an operational prototype for testing.
- Testing – Involves comparing the prototype with the performance standards of the domain experts. Testing is an iterative process.

The software development cycle described by DOD-STD-2167A is not descriptive of the development process for ESs. The development process for ESs is a much more iterative process than that of conventional software. Also, the five phases of development are not independent and clear-cut, but describe a knowledge acquisition process. To accommodate knowledge-based systems, a new development cycle needs to be defined and incorporated into the appropriate standards (Carlton 1987).

5.3 ARTIFICIAL INTELLIGENCE STANDARDS.

Since AI is a relatively new technology, a lack of standardization exists. Standards relating to AI are only beginning to emerge. This generates difficulties, especially from the developer's perspective. Adherence to standards can have many benefits for developers, including interoperability through common interfaces, as well as product portability. Additionally, lower overall life-cycle costs can be realized through increased reliability and maintainability. Areas where standards could benefit the technology include knowledge representation, ES shells, and AI language standardization.

5.3.1 Expert System Shells.

Run-time and development versions for ESs differ with respect to the knowledge and data that are incorporated in the system. Warn (1987) identifies areas where differences are noted:

- Domain knowledge
- Support knowledge

- Knowledge base
- Simulated current situation data

The selection of an ES shell for development is often based upon rapid development capability and support. However, when the development cycle is completed, the development interfaces of the ES shell are not required for the run-time ES. Moreover, if a developer wishes to use a different inference engine, there may be compatibility problems.

Highlighting the lack of standards for ES shells, Warn states:

> A major drawback of the current state of proliferation of in-house and commercially available expert system shells is the lack of standardization. The result is that an expert system developed on one shell cannot, except in rare cases, be ported to a different supplier's shell.

To overcome this drawback, he suggests a potential solution: the Common Run-Time Inference Engine (CRTIE). Figure 5.3-1 shows the interfaces suggested for CRTIE. The application of various support tools is made possible by interface standardization. However, as newer and improved tools for ESs become available, the need for a common interface provision may diminish.

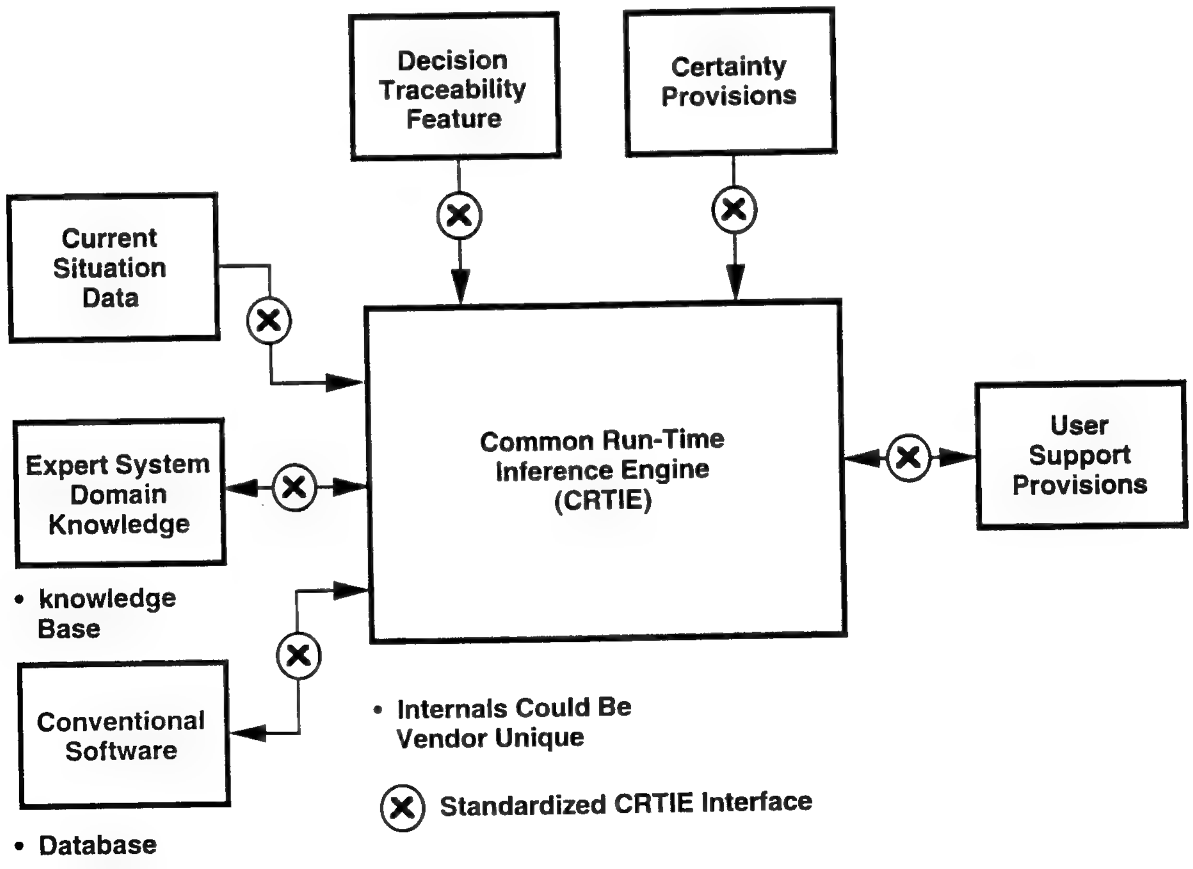

Decision Traceability Feature
Certainty Provisions
Current Situation Data
Common Run-Time Inference Engine (CRTIE)
User Support Provisions
Expert System Domain Knowledge
• knowledge Base
Conventional Software
• Database
• Internals Could Be Vendor Unique
Standardized CRTIE Interface

FIGURE 5.3-1. COMMON RUN-TIME INFERENCE ENGINE
(Warn 1987)

5.3.2 Language Standardization.

In addition to the lack of shell standardization, development languages that are nonstandard can increase costs. Common LISP was developed in an attempt to provide a standard AI programming language. The immediate benefit of standardizing a language would be the portability of the source code across the various platforms that support it.

5.3.3 Standardizing Knowledge Representation.

Creation of a knowledge representation standard could be useful in applying analysis tools. A working group of the IEEE was formed to develop P1252, Frame-based Knowledge Representation Standard. No draft has been released yet.

5.3.4 Artificial Intelligence-Expert System Tie to Automatic Test Equipment.

Artificial Intelligence-Expert System Tie to Automatic Test Equipment is a standard under development by the IEEE. It will define an architecture for AI-based ATE in addition to describing a set of associated models. Several issues will be addressed including model base specifications, data and knowledge management, data and knowledge interchange, and various services, such as human, program, and network (Sheppard and Simpson 1992).

5.4 CONVENTIONAL SOFTWARE VERIFICATION AND VALIDATION.

This section examines techniques, methods, and issues relating to the V&V of conventional software-based systems. A review of these issues is necessary to provide the background and framework for the application of these techniques to AI-based systems. Once the issues are understood for conventional software, they can be applied to other types of systems and tailored for specific characteristics of those systems.

Although this section deals with software issues, it is important to note that the processes described in this section can and should be applied to hardware, software, and systems. New techniques may be necessary for ensuring safety goals for today's highly integrated and complex systems. Each component, as well as the total system, needs to be subjected to scrutiny that yields the required assurance level. Today's highly integrated avionics, using silicon devices with millions of transistors in a single package, require the application of V&V techniques specifically designed for this level of complexity, just as ESs require unique V&V techniques.

5.4.1 Verification and Validation.

5.4.1.1 Verification.

There is no exclusively correct set of software statements for implementing solutions in software-based systems, since there are virtually an infinite number of different ways to represent solutions to system problems. Since software solutions are used in systems where failure can result in loss of life and property, methods are needed to demonstrate that the particular software solution chosen to solve a system problem is correct and safe. This is one of the functions of the verification process.

According to the glossary of the Digital Systems Validation Handbook-Volume II, verification is defined as:

> The act of reviewing, inspecting, testing, checking, auditing, or otherwise establishing and documenting whether or not items, processes, services, or documents conform to specified requirements.

5.4.1.1.1 Verification and Documentation.

One of the important ingredients in the process of verification is documentation. According to Naser (1991), "Documentation creates a traceable and systematic approach to the development of a product." Traceability is important since it demonstrates the flow and development of concepts and design elements from the requirements documents through the final product.

When a system is well documented, it can be understood by knowledgeable persons other than the developers. Persons other than the developers are often used in the verification process to analyze the product from a different perspective. This activity is referred to as Independent Verification and Validation (IV&V).

Figure 5.4-1 shows a typical DOD development cycle, along with the associated products, for a large-scale conventional software system. Problems in the development process often appear as the result of the transition from one step to the next. Part of the verification process deals with determining that a complete and accurate translation has been performed by the developers.

A requirements document describes the system at the highest level. It goes through many iterations of refinement during development. It also requires maintenance throughout the product life cycle.

Traceability depends on the completeness and accuracy of the requirements document. The requirements document is viewed as a set of external performance goals. Testing explicitly confirms that these product or system performance goals can be met. Testing against the requirements is both desirable and necessary (Naser 1991).

5.4.1.2 Validation.

The Digital Systems Validation Handbook describes validation as:

> Demonstration and authentication that a final product operates in all modes and performs consistently and successfully under all actual operational and environmental conditions founded upon conformance to the applicable specifications.

Validation is performed on a system to ensure that the requirements and the end product, operating in its intended environment, have a direct correspondence. Validation is a process that is composed of a number of activities. The purpose is to demonstrate that a piece of software is designed correctly and that it is the correct or right design. The functionality, performance, and device interface are given in the requirements document. If the product does not meet these criteria, then the deficiency can be traced back to the cause through each of the step-wise refinements of the development process.

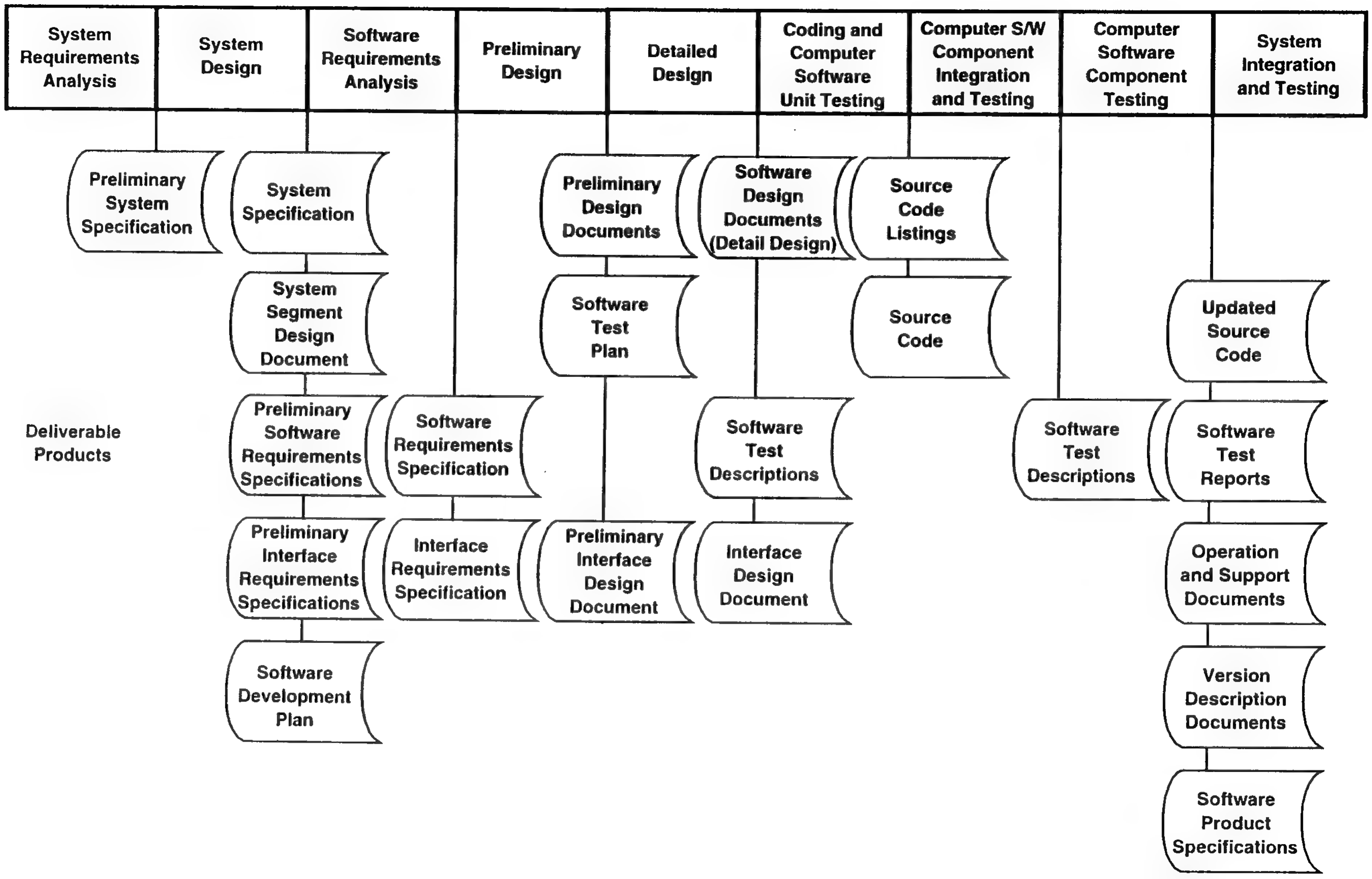

System Requirements Analysis
System Design
Software Requirements Analysis
Preliminary Design
Detailed Design
Coding and Computer Software Unit Testing
Computer S/W Component Integration and Testing
Computer Software Component Testing
System Integration and Testing
Deliverable Products
Preliminary System Specification
System Specification
System Segment Design Document
Preliminary Software Requirements Specifications
Preliminary Interface Requirements Specifications
Software Development Plan
Software Requirements Specification
Interface Requirements Specification
Preliminary Design Documents
Software Test Plan
Preliminary Interface Design Document
Software Design Documents (Detail Design)
Software Test Descriptions
Interface Design Document
Source Code Listings
Source Code
Software Test Descriptions
Updated Source Code
Software Test Reports
Operation and Support Documents
Version Description Documents
Software Product Specifications

FIGURE 5.4-1. SYSTEM DEVELOPMENT CYCLE (DOD-STD-2167A)

One may deduce at this point that errors found in the validation phase are the most costly to correct. Since development proceeds in a step-wise manner, the correctness of each step is essential to avoiding mistakes that will ripple down into the end product, where they are consequently discovered.

5.4.1.3 Aspects of Verification and Validation.

5.4.1.3.1 Correctness.

There are several different aspects of correctness that require examination. These include functional, safety, user-interface, resource consumption, and utility correctness. Consistency, completeness, and termination are also aspects of correctness, but are examined separately.

5.4.1.3.1.1 Functional Correctness.

Functional correctness is concerned with how a particular piece of software behaves, based on various system inputs. This behavior is defined by the system documentation. In general, it is not possible to test all input combinations completely. Therefore, a number of techniques and rationales exist for generating tests that are the most beneficial for demonstrating functional correctness of the subject software. Test methods to demonstrate functional correctness check to see that the program generates the correct system output for any input. Testing of all cases is generally not possible, therefore only certain sets of test cases are examined. The preferred set of tests include those that are most likely to occur in normal operation.

There are several methods used for test case selection, some of which follow:

- Cause-effect graphing – Consists of mapping the input and output paths, where the conditions are expressed in Boolean form. Intermediate steps are also shown, yielding an accurate representation of the level of abstraction of the software.

- Boundary testing – A technique used to test input boundaries defined in the software. Testing the response using inputs near the boundary provides significant data on the correctness of the software.

- Attribute-based test case selection – Requires that test cases are selected based on software characteristics. For instance, a specific test case may be designed for a complex part of the software, while a simpler part may be passed over. Criticality, as well as program size, are attributes used for test case selection.

- Random testing – A technique for test case selection. As the name suggests, the method of selection for this technique is entirely random. Using this method, no rationale is required for selection of test cases.

5.4.1.3.1.2 Safety Correctness.

Safety correctness deals with showing that software never reaches a state that poses a danger to man or machine. With the increasing complexity of systems, this attribute may be easier to demonstrate than functional correctness.

Safety correctness testing is concerned with picking tests that are designed to generate unsafe states in the software. Knowledge of the controlled device is necessary to identify unsafe software states. Various combinations or sequences of inputs are exercised while the response is examined for unsafe conditions.

5.4.1.3.1.3 User-Interface Correctness.

User-interface correctness is concerned with the end-user and human factors aspects of the software interface. Some of the human factors aspects that are considered include ambiguity, correctness, and "user-friendliness."

5.4.1.3.1.4 Resource Consumption Correctness.

Resource consumption correctness demonstrates that the software operates efficiently within its environment. It focuses on performance issues, such as general hardware architecture, disk utilization, communication channel bandwidth, and meeting specified timing constraints. This method is essentially a performance test to check whether or not a system operates efficiently for its intended application.

5.4.1.3.1.5 Utility Correctness.

Utility correctness focuses on the usefulness of the system. It helps to determine if it is the right system, if it does the right things, if it does what the customer wants, and if it meets system requirements.

5.4.1.3.2 Consistency.

Consistency demonstrates that the software follows a predictable path. Two forms of consistency are examined: internal and external. External consistency demonstrates that when a particular event occurs, an associated routine is exercised. External consistency also considers the user interface. The look and feel should be consistent. Internal consistency involves showing that all parts of an integrated software module are consistent.

5.4.1.3.3 Completeness.

Completeness involves demonstrating that nothing is missing from the program. Completeness seeks to demonstrate properties, such as the following:

- The software accepts and acts upon all data.
- The software generates all required outputs.
- The software performs all required actions.

There are no algorithms that can be exercised to demonstrate completeness. Sufficient completeness should be demonstrated to show that the requirements are satisfied.

5.4.1.3.4 Termination.

Another aspect of correctness is the demonstration of termination. This characteristic ensures that a program will always end. Demonstration of termination is done by showing that all looping structures within the program terminate (French and Hamilton 1992).

5.4.2 Verification and Validation Testing Phases.

Testing is beneficial for several reasons:

- Both major and minor errors can be identified.
- Errors introduced into the system can be reduced.
- Confidence in the product can be increased.

This section discusses classes of testing performed on software, including static and dynamic testing. Testing methods are also discussed.

5.4.2.1 Static Testing.

While dynamic testing involves running the software in the target environment, static testing involves analyzing the software by methods other than execution of the code. Static testing is done either by software review or by using tools to analyze the software. A detailed examination of the equations contained in the software is performed, or other checks are made against the specification.

The static testing method is the most cost effective since it occurs early in the development phase. As software development proceeds from the requirements phase to the system test phase, the cost of fixing errors increases drastically. Errors identified and corrected in the requirements phase have the smallest impact on the overall development. This is why verification needs to start at the requirements phase and be as thorough as possible.

Some of the information that can be obtained by static analysis includes the following:

- Uninitialized variables
- Variables set, but not used
- Isolated code segments not executed under any set of input data
- Cross-reference maps
- Analysis of identifier usage
- Statistics about source code statements

The timing of software as it is being exercised in its environment is important to checking correctness. Software timing cannot be determined with static testing.

5.4.2.2 Dynamic Testing.

Dynamic testing is testing of the software in its intended operational (target) environment. Dynamic testing has two parts: system testing and unit/integration testing. This testing method is the least cost effective since it is performed later in the development phase. Errors identified and corrected in this phase of testing have a larger impact on the overall development cost and schedule. Detailed verification at the requirements phase can greatly reduce the amount of rework that may otherwise be performed in later stages of the development process.

Following are some system characteristics can be examined and tested at this point:

- Functional performance to evaluate software correctness
- Error handling
- Interaction
- Initialization
- Timing

5.4.2.2.1 Integration Testing.

Integration testing is referred to as a "white box" testing technique (French and Hamilton 1992). It is dynamic and focuses on I/O response classes. Integration testing can deal with multiple levels of abstraction, whereas system testing deals with only one. This technique is concerned with the correctness of the unit.

Integration testing depends on modularity of the software. Modularity implies that the design and development process can be simplified by breaking a large and complex task into smaller but more manageable pieces. Each piece can be handled by a different person or group, and can, after thorough testing, be integrated with the other pieces. Modularity is a benefit to verification in that when changes are made, only the module that is changed needs to be verified.

5.4.2.2.2 System Test.

The system test is the last test to be performed. It is the final step in the development process. Demonstrating correctness at this stage does not require knowledge of the internals of the Unit Under Test (UUT). It is simply treated as a "black box." This black box is tested to see that the behavior it exhibits is correct. The responses of the black box are based on the original requirements that were established with the customer. The contents of the box are, at this point, irrelevant and not the subject of any investigation. Independent verification is desirable, since knowledge of the internals are irrelevant during system test and may be a stumbling block to testers.

It is generally accepted that complete testing of software is not possible. This is due to the fact that, for any given system, there are too many I/O responses to be tested. Complete testing of a system is not done, and, therefore, it is unknown whether or not the software in the black box is completely correct.

To devise practical limits on the number of I/O responses to be checked during testing, the software in the black box can be divided into categories based on the classification of the I/O responses. These individual categories can then be treated as separate units and tested as though each were its own separate box.

5.4.2.3 General Testing Methods.

The three general methods of testing are prototyping, regression testing, and IV&V. Prototyping involves building a scaled version of the real system. This is then used to evaluate the operation of the system. Regression testing involves testing older software that has undergone modification. It ensures that the modified portions of software do not affect other portions that were not changed. IV&V is a testing method that calls for an organization not involved in a product's development to perform the V&V. This method seeks to avoid prejudice towards that particular product on the part of the developers.

5.5 EXPERT SYSTEM SOFTWARE VERIFICATION AND VALIDATION.

V&V is an essential activity for any system where software criticality is an issue. V&V is as important for ESs as it is for any software system. The adaptation of existing V&V methodologies to ESs requires an appreciation of the risks inherent in the particular implementation just as it does for conventional software. The variation in ES applications and implementation covers a wide range just as it does for conventional software. The V&V testing should be planned accordingly. In many cases, an ES will require less effort to validate than a conventional software application of comparable complexity. Methodologies for V&V of ESs is an area of ongoing research.

Conventional V&V techniques can and should be used for parts of an ES, such as the inference engine; particularly if the inference engine is implemented with conventional software. However, the knowledge base is substantially different from conventional software implementations requiring the application of new V&V techniques.

Various types of ESs exist, differing in the knowledge source and knowledge type. V&V should be tailored to the type of system under development. Naser (1991) identifies six different types of ESs:

- Type I – The simple system. The knowledge base for this system is already verified and validated, never increases or decreases, and can be partitioned. The system is one of finite size and concept.

- Type II – Similar to I, but has uncertainty handling.

- Type III – A simple system, but the knowledge base is derived from the expert in an iterative fashion and therefore requires V&V of the knowledge base.

- Type IV – Similar to III, but has uncertainty handling.

- Type V – The complex ES. Nonmonotonic reasoning and large unfactorable knowledge base, learning capability, and multiple knowledge bases with potentially conflicting heuristics.

- Type VI – Similar to V, but has uncertainty handling.

ES applications that are limited in scope, having fixed and easily verifiable knowledge bases, should require significantly less V&V effort than those dealing with pilot intent interpretation. Software modules of the PA need to be subjected to more intense V&V because of the scope of the PA, the information the PA has as input (such as pilot intent), and the capability of the PA to exert control over other aircraft systems. A diverter ES, on the other hand, is much more limited in scope, deals with trustworthy data at the input (such as the AIM and FARs), and offers advice without exerting control over other systems.

In general, ES applications proposed for commercial aircraft have limited complexity. However, this report presents V&V techniques that address simple, as well as complex ESs. One of the tasks of the certification specialist is to determine the required level of testing based on the proposed application of the ES.

5.5.1 Contrasting Expert Systems and Conventional Software.

The types of problems that ESs are used to solve are highly complex and too difficult to solve by conventional computer methods.

For these problems, a solution exists in the mind of the expert. This solution is derived by application of the expert's domain knowledge. This knowledge is then translated by the knowledge engineer into terms that a computer can understand. The manner by which problems are represented and solved constitutes the fundamental differences between ESs and conventional software.

5.5.1.1 The Software Development Cycle.

While many techniques that are used for V&V of conventional software can be applied to ESs, there remain areas where new techniques are required. Some of the differences are examined in the following sections. In spite of the differences between ES and conventional software development, ESs demonstrate a recognizable life cycle. There are well defined steps within this cycle to which V&V techniques can be applied.

Figure 5.5-1 depicts a development cycle for conventional software. It is referred to as the "waterfall" model, since each step is completed before the next step is begun, and work proceeds progressively from left to right, without return paths to prior steps.

As each step is completed, additional products are generated. For example, when the Preliminary Design Phase is completed, the Preliminary Software Design Document(s), Software Test Plan, and Preliminary Interface Design Document result.

Specifications and design documents are as necessary for ES development as they are for conventional software. Performance goals and functionality are measured against these documents. To implement any verification technique, an objective for a project must exist and be well documented. Without an objective, there is little help for the developer and no hope for the verification effort. Traceability requirements start with concrete measurable objectives. From this point, further refinements are made until the requirements are well-defined.

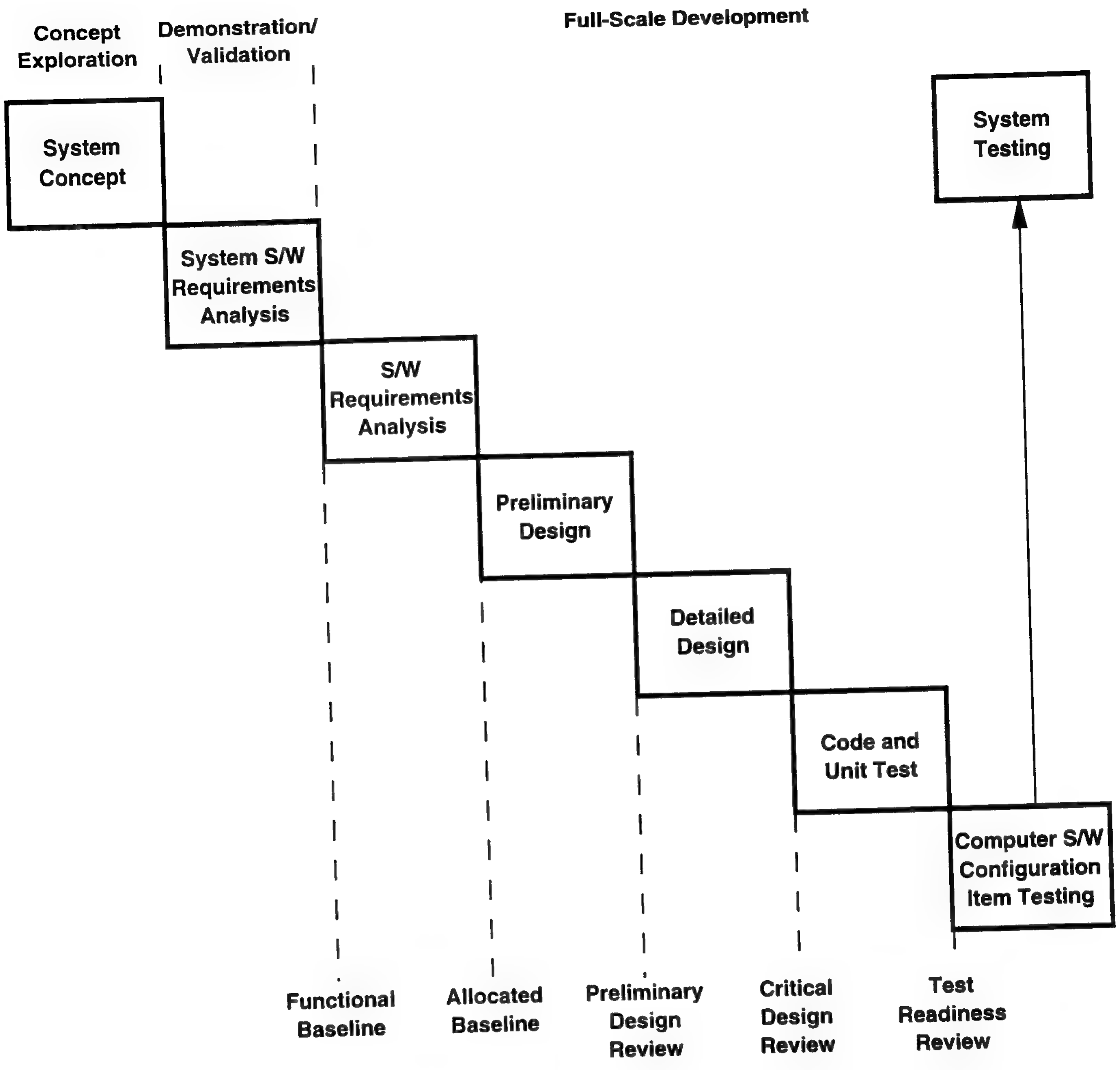


FIGURE 5.5-1. BASIC WATERFALL MODEL OF SOFTWARE DEVELOPMENT

As the system is developed, document maintenance is required. ES requirements often emerge throughout the development cycle, necessitating incremental build and test cycles. According to Naser (1991):

> Testing the prototype reveals deficiencies in performance, suggests holes in the knowledge base, and stimulates another round of knowledge-building, coding, and testing.

This cyclic development for the ES is seen in the diagram in figure 5.5-2.

5.5.1.1.1 Development Complexity.

The power that an ES shell environment offers the developer and the user is often employed in developing applications far more complex than would ever be contemplated using a conventional software approach. This is usually manifested by a large number (many hundreds) of rules in an attempt to model a complex process with extremely high fidelity.

If the application is so difficult that only a powerful approach can hope to succeed, the problem of V&V comes from the scope of the application, not from the approach. A program such as the PA has very difficult goals, regardless of the development technique. An ES approach to the PA reduces the complexity of the overall task.

As with the verification of any critical and complex software application, the V&V approach must include an assessment of the risks to correct performance and direct attention to minimizing exposure to these risks. A major risk that is avoided with an ES approach is the integrity of the logic applied to the rules in the application. The V&V effort should also focus on the question of the integrity and fidelity of the rules relative to the problem domain. This effort is tenable, even when there are a large number of rules. This is because the rules are in the form of English-like statements, and, more importantly, the logic taken by the application on test cases can be examined minutely.

The complexity of the internal interactions of a software system depends not so much on the implementation technique, but on how the developer has applied the technique. Programs written in C can be as cryptic as the developer chooses. ES internal interactions that are complex need to receive greater attention in verification activity so that all consequences are known. It is possible to design ESs that have a more structured appearance and, therefore, facilitate testing.

In some cases, the inference engine may be implemented using LISP. When this is the case, the developer will be required to implement a garbage collection scheme.

Table 5.5-1 summarizes some of the major differences between ESs and conventional software.

5.5.1.1.2 Determination of Correctness.

It is difficult to determine the correctness of a system that generates solutions based on the heuristics of an expert. The domain expert is the only person qualified to comment on the correctness of the ES. Knowledge base verification will be a key focal point of certification activity for critical software.

5.5.1.1.3 Expert System Shells.

As is the case with conventional software compilers, ES shells may contain misleading documentation or exhibit faulty operation. Schultz and Geissman[1] (1991) point out that these potential problems may ultimately lead to incorrect operation of the developed system if they go unnoticed. To develop reliable systems, the knowledge base, database, and user-generated code, as well as the developer's tools need to be validated.

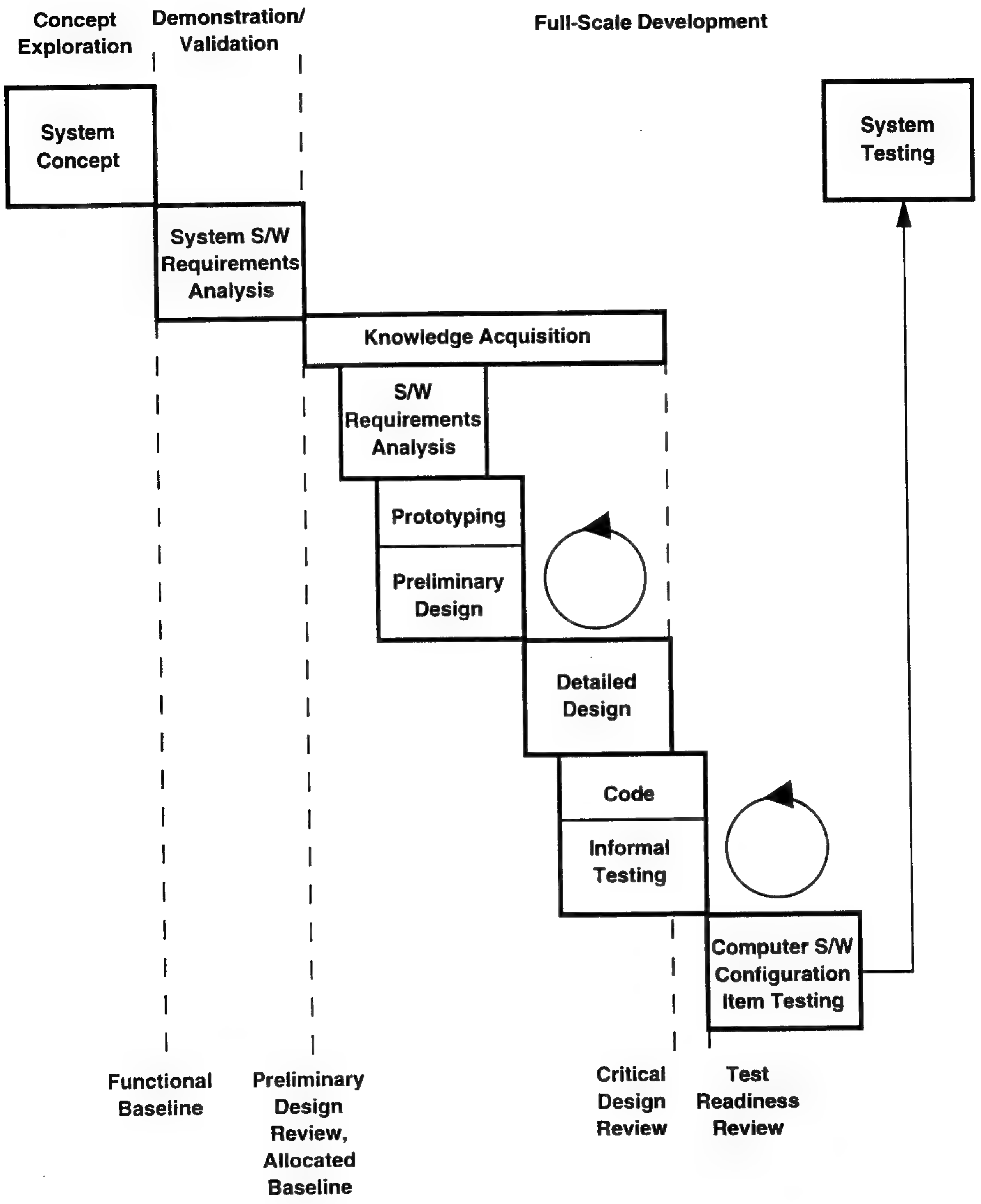


FIGURE 5.5-2. DEVELOPMENT CYCLE DIAGRAM FOR EXPERT SYSTEM
(Carlton 1987)

TABLE 5.5-1. EXPERT SYSTEM AND CONVENTIONAL SOFTWARE DIFFERENCE SUMMARY

Expert Systems	Conventional Software
• Context driven, nonprocedural	• Sequential execution, procedural
• High-level, abstract data representation	• User-generated data representation
• More complex internal interactions	• More decisions
• May require garbage collection	• More control structures
• Rule interaction	• Program dependent internal interaction
• Intelligent control behavior	• Complexity
• Iterative development	• Stepwise development

Paradigms (such as chaining) implemented in the tool need to be identified. The tool then needs to be examined to see if the paradigms are carried out correctly. The importance of a validated ES shell cannot be over-emphasized, since the attributes and capabilities of a system are influenced and limited by the weakest link of that system. All parameters of the tools that affect the operation or timing of the end product must be known and specified.

The easiest kinds of ESs to validate are those developed with an established commercial ES shell. These environments feature built-in rule editing and data management, graphics interfacing, and accessibility by direct incorporation of user written C code. They also support justification of answers by displaying on request the chain of rules by which they have come to a conclusion. Since the language that these ESs execute is in the form of English-like rules, the process of verifying the correctness of the program logic is greatly simplified. Both the accessibility of the language and the justification feature make it possible to achieve a level of confidence that an ES application developed and supported by such an ES shell does what was intended and with high reliability.

5.5.1.1.4 Inference Engine Verification and Validation.

In the past, ES users created their own ESs using conventional software. In so doing, they also programmed the inference engine. While the availability of commercial ES shells has greatly reduced the number of custom ESs, when this is the case, and the application's reliability is critical, the V&V effort should also extend to the inference engine. This is all the more true if the code and the knowledge base for the inference engine are to be maintained by the customer. Since ES shells contain inference engines, using a commercially available shell for system development can reduce, if not eliminate, the need for conventional programming. Also, the time to develop a system can be reduced by using a commercially available shell.

At a fundamental level, an inference engine matches antecedents of some rules with the consequences of others in what is interpreted as a logical sequence of reasoning. The testing of an inference engine involves developing a suite of rules whose behavior is well understood and encoding them into the form required by the inference engine. The test would be designed to show that, at a high level of confidence, all the possible connections between rules are explored in arriving at the conclusion to tests. The fact that the same answers are received by the inference engine under study as are received by the trusted inference engine should give high confidence that the inference engine is trustworthy.

5.5.1.1.5 Flow Control.

Knowledge-based systems are more context or data driven than systems developed with conventional software. Testing is more time consuming when data driven software is used due to an increase in the number of branches that require inspection. For verification purposes, it is easier if flow is determined at design time, as with conventional software.

5.5.1.1.6 Implementation Language.

Conventional software is developed using procedural languages such as Ada, C, Pascal, or FORTRAN. Knowledge-based systems use a combination of both non-procedural and procedural languages.

5.5.1.1.7 Deterministic Behavior.

The underlying processes of a knowledge-based system are often complex and difficult to predict, but most of the time the system remains theoretically deterministic. The main exception is when learning is designed into the system. Hammer (1991) indicates that it would be difficult to verify that a system incorporating learning behaves safely at all times. Deterministic behavior must be demonstrated for ESs and conventional software systems.

5.5.1.1.8 Use of Dynamic Memory.

Computer programs often allocate memory from a "heap" when memory resources are required during the execution of various software procedures. For verified software, there should exist a statically determined upper bound, so that a routine requiring memory will never lack this resource. This memory allocation should be determined during the design phase.

5.5.1.1.9 Domain Knowledge Review.

In order for knowledge to be utilized by a computer, it is transformed extensively. A detailed review for correctness should be performed by the domain expert before translation. After undergoing transformation for use in a computer system, the original knowledge is in concealed form and the transformation process and product should be carefully examined.

5.5.1.1.10 Verification of Knowledge-Based Requirements.

Requirements specify system behavior. Understanding intelligence well enough to specify its interactions in a requirement document is difficult. Intelligent behavior consists of adaptivity, not simply complexity.

Therefore, verification of a knowledge-based system should involve the testing of intelligent control behavior.

5.5.1.1.11 Program Structure.

It has been suggested that another difference between ESs and conventional software is that ESs are unstructured while conventional software is structured. While certain High-Order Languages (HOLs), such as Pascal and C, are referred to as structured languages, the degree to which a program reflects this characteristic is dependent upon the programmer exercising it. Software whose native language is not structured can be implemented in a structured manner by the programmer. What matters for ESs, as for conventional software, are the techniques used to implement a software application.

A related issue is the modularity of the system. With conventional software, tracing system development from the requirements to the code demands the use of a hierarchical structure. A technique called structured design is used. When this method is followed, procedures are generated in a top-down incremental fashion. In the past, rule-based ESs have lacked the hierarchical structure of conventional software. Rules were intermixed with each other and with control rules since no standard techniques or development tools existed to assist in achieving the goal of modularity. Modern design tools, however, allow rules to be categorized into classes by the designer. The ability to express the high levels of abstraction made possible with ESs allows designers to structure the knowledge base, along the most natural divisions that exist, as part of the knowledge.

5.5.2 Design Issues in Verification and Validation of Expert Systems.

Techniques used (or not used) in the development of AI systems can greatly enhance or detract from the ease with which a product may be subjected to V&V. Following are suggestions to assist this process (Hammer 1991):

- Verification with respect to knowledge base design
  - Checklist reviews for quality of design
  - Requirements to design mapping
  - Test cases for exercising the design
- Characteristics of a design which make verification easier
  - Design using components of limited size for checking
  - Hierarchical top-down design approach
  - Clear interfaces between components

Some useful advice is given by Schultz and Geissman[1] (1991) on concrete ways to assist the verification process. Clear and concise documentation on the following is recommended:

- Generic task or tasks
- Triggers that begin inferencing
- Goals
- Paradigms used

- Subproblems and their interrelationships and transitions
- External interface requirements
- All relevant domain knowledge
- Procedures for deriving knowledge
- A high-level statement of generalized inferencing paths

5.5.3 Specification and Development Differences.

Referencing DOD-STD-2167A, Schultz and Geissman[1] (1991) state:

> The verifier of an Expert System will probably be frustrated by the lack of intermediate products to check off. Standards like DOD-STD-2167A speak of system and software requirements and several levels of design before code is reached. Standards specify what each level of documentation should cover and the methodology for their production.

Compared with conventional software, the number of design stages in an ES development project will be fewer due to the high level of expressiveness. The language of the ES operates at a much higher level than that of conventional software and may appear more like a requirements document than code. This is a benefit for ESs, since the verification process is concerned with the details of fewer intermediate steps. The number of associated documents will also be fewer than for conventional software.

In light of the lack of intermediate products, Schultz and Geissman[1] (1991) suggest that the following are appropriate by-products of ES development and would establish a baseline for verification:

- ES requirements derived from a prototype generation
- A design document
- The code or knowledge base

According to Rouse, Geddes, and Hammer (1990), ES development work performed for portions of the PA program used a technique known as rapid prototyping. This method allowed shorter prototyping cycles than are typical for such development. Additionally, the development process was accelerated through a modification in the software documentation process. The requirement of extensive software documenting before writing the software was set aside to allow many rapid prototyping cycles. Details of the software were delivered after completion of each prototype cycle.

5.5.3.1 Vaguely Specified Systems.

V&V of ESs is difficult when there is uncertainty that the rules in the knowledge base properly represent the best approach to a problem. This situation arises when the problem domain is not fully understood. When a knowledge base is developed by interviewing an expert who is unwilling or unable to define the processes used in solving a problem, it is up to a knowledge engineer to translate that knowledge into rules. Even with a cooperative and articulate expert, the process of refining a rule base can falter when the effort to make explicit expertise reaches the intuition level. It is frequently at the intuitive level that the performance of an expert is distinguished from that of a rule-follower.

Challenges to the V&V effort result when the requirements for the system, and specifications to which the system was developed, lack detail. The performance of the knowledge base must then be tested empirically, with the proof involving the approval of experts in the problem domain. The problems of verification are not insurmountable if the inference engine is verified independently and there is a large number of tests, perhaps developed by domain experts, for which the answers are independently known.

5.5.4 Implementation Differences.

Conventional software and ESs are similar in that both consist of software which executes on a digital computer. There are associated inputs and outputs for any software system. An AI-based system or ES may even use a conventional HOL. Parts of the ES are typically implemented using conventional software.

Normally, the ES user develops ES applications through the use of tools. A common tool is the ES shell. The ES shell gives developers the capability to create systems without using procedural languages. Additionally, the problems that are solved using ESs have different characteristics. To be effective, the V&V approach and activity should be tailored to address these differences.

Procedural programs use a sequence of executed statements which are followed easily. The next statement to be executed in a sequence is based on the value of a defined set of variables. With non-procedural languages, the execution flow is not determined as easily. This is due to the characteristics of the inference engine used to control execution. Data and control are partitioned in ESs. The inference engine determines what statement is executed next by taking into account the basic state of the program together with the data. Partitioning for the knowledge base and built-in traceability tools for the shell assist developers in generating systems where the execution flow can be traced and explained easily.

Tool performance limitations may become a factor in the development process for ESs, as also may be the case for conventional software. There are performance variables associated with tools that are not apparent to the developers until actual real-time operation of the program.

All software is developed iteratively. ESs, however, require greater amounts of iteration in the development process since development involves extracting information from the mind of domain experts. Many of the design assumptions made during early stages of development will undergo changes during the ES development cycle. Therefore, an incremental approach to development is viewed as necessary.

It must be said, however, that current conventional software development allows a far greater degree of iteration than in the past. New techniques and tools, such as Object-Oriented programming, applied to the development of conventional software systems have allowed rapid and iterative development cycles which former methods could not support.

5.5.5 Verification and Validation Techniques for Expert Systems.

Conventional V&V techniques can be applied to many parts of the ES development process. These parts are identified in table 5.5-2. The knowledge base is not included in the table. It is a chief concern and represents a difficult area for V&V techniques.

TABLE 5.5-2. EXPERT SYSTEM PARTS FOR CONVENTIONAL VERIFICATION AND VALIDATION COVERAGE

Expert System Components
• Implementation hardware
• Expert System shell – Inference engine – Knowledge representation schema – Control options – Utilities
• Customized user interface
• Special methods

There are a number of techniques used for V&V. No single technique will be efficient for all applications. Different types of errors and characteristics are tested with each technique. Two general approaches exist. One examines characteristics of the ES implementation. The other examines whether or not the solution fits, or solves, the problem. This approach is implementation independent. The former method is called a "clear box" technique, as opposed to the black box technique. Both techniques are essential.

V&V methods are sometimes classified into two groups: qualitative and quantitative. A qualitative method applies "subjective comparisons of performance" to the ES. A quantitative method uses statistical techniques for comparison of ES performance against test cases or experts.

O'Keefe, Balci, and Smith (1991) divide the quantitative validation method into two categories: one uses a confidence interval and the other uses a hypothesis test. The confidence interval is compared subjectively with an acceptable performance range, whereas the hypothesis test is a formal test that compares measurements against a predetermined acceptable performance range.

A conventional software subroutine that manipulates variables passed to it based upon a mathematical procedure, will produce a solution for which only a single value or set of values is the correct result. Hence, correctness is easily tested. A difficulty often arises with ESs because there may be more than one correct solution to a problem. What constitutes an acceptable or unacceptable solution from the system should be defined clearly before verification begins (Schultz and Geissman[1] 1991).

Following are the qualitative techniques (O'Keefe, Balci, and Smith 1991):

- Face validation – Compares ES performance against that of a team of experts. A judgement is made against some prescribed performance criteria.

- Predictive validation – Uses historical test cases along with known results or human performance measures. An evaluation is made based on this comparison.

- Turing tests – Uses a blind evaluation technique. Experts review the conclusions of an ES for a given situation. These conclusions are evaluated without knowledge of their origin. This method serves to eliminate any human prejudice for or against the ES.

- Field tests – Places the burden of proof for error identification upon the user. A prototype system is installed and users exercise it and report errors that they find. This method is used for noncritical applications only.

- Subsystem validation – Requires that an ES be partitioned into a number of subsystems. As these subsystems are developed they are validated separately. This simplifies some aspects of validation, but does not guarantee the validation of the entire system.

- Sensitivity analysis – Determines if the response of the ES to changes at one of the inputs is appropriate. All inputs are held constant, except for the input under consideration.

- Visual interaction – Provides the expert with the ability to interact with the system through a visual animation of the ES. This method can provide a basic platform for some of the validation techniques described above.

Table 5.5-3 summarizes some V&V techniques for ESs (Naser 1991).

5.5.5.1 Testing the Implementation.

Testing the implementation involves checking interactions to see if they are intended and correct. In systems where there is a large amount of interaction, this can be a difficult task. Several techniques for implementation testing are examined in this section. These techniques do not identify errors, but provide an indication that an error may exist. It is then up to the programmer or domain expert to determine if the error actually exists.

Testing ESs can be similar to testing conventional software. Testing conventional software can be performed by exercising decision branches, checking looping conditions, and testing boundary conditions. Applied to ESs these techniques could be mirrored by testing every rule or sequence of rules, if the rule base is not too large, and exercising rules based on the rule conditions. Additionally, just as conventional software is tested for quality of implementation, ESs should be checked for problem indicators. These include obscure constructs, overlapping rules, incongruous rules, incomplete rule coverage, and infinite rule loops (Hammer 1991).

5.5.5.1.1 Testing Rule Consistency.

Rule consistency checks for the absence of undesirable characteristics in ES implementations. These characteristics include the following:

- Dead-ends – A rule that has no effect on any other rules.

TABLE 5.5-3. EXPERT SYSTEM VERIFICATION AND VALIDATION TECHNIQUES

- Formalize the requirements in a specification document.
- Verify that the requirements specification captures the requirements, including those for system usability.
- Verify that the requirements specification is implemented in the system design.
- Verify that the design is maintainable.
- Verify adequacy of knowledge and accuracy of knowledge presentation.
- Verify that the requirements are met for all interfaces of the system.
- Verify rule completeness and consistency.
- Conduct comprehensive shakedown testing, exercising as much of the system as possible.
- Verify system usability.
- Conduct tests for special cases and boundary conditions.
- Verify that the system is useful and fulfills its design purpose.

- Unreachable rules – Rules that are not affected by any other rule.
- Circular rules – Rules that have the potential to repeat endlessly.
- Conflicting rules – Rules that contradict each other and perform actions which are in opposition.
- Redundant rules – Rules that contain the same information and perform some of the same actions as other rules. Although redundant rules may not actually produce errors, they should be identified and checked.

5.5.5.1.2 Testing Data Consistency.

Data consistency involves checking that data or facts are used as defined. An operation that writes a 16-bit value into a variable defined as only 8 bits would be detected by such a check.

5.5.5.1.3 Sensitivity Analysis.

Sensitivity analysis is an examination of how one variable may influence other variables. This type of analysis is useful in classifying problem solutions (French and Hamilton 1992).

5.5.5.1.4 Structural Testing.

Structural testing is performed to ensure that a set of tests is comprehensive in coverage. This type of testing can be adapted to ESs by defining coverage to include rules, frames, and frame demons (procedures activated by changing or accessing frame data). When changes are made to the knowledge base, the coverage again must be determined and new tests created and executed.

5.5.5.1.5 Specification-Directed Analysis.

Specification-directed analysis is a method for demonstrating that a program is traceable to the specification from which it was produced. This is performed by first arriving at a method which identifies and records specifications. These identified specification items are then checked against the program for correspondence.

In addition, the knowledge represented needs to be adequate for the intended use of the system. This requirement helps define test cases so that test coverage will be complete. When incorrect responses are found in the system, it should not be difficult to trace the incorrect response to its origin.

5.5.5.2 Test Bias Elimination.

Biases in testing should be avoided, where possible. One particular bias is the use of test cases that were used during the development process. Presumably, the system was designed in such a way that it responds favorably to these tests. For completeness, some of these tests should be used to ensure that they still work. However, testing should focus on areas in which development testing has not been performed.

As with conventional software, validation and testing should be performed by a team independent of the development process to avoid prejudice.

5.5.5.3 Test Coverage.

Reasonable cross-sections of test coverage should be utilized. It is questionable whether situations that occur only twice in every 100 cases should receive greater scrutiny than 2 percent of the total testing effort. This scrutiny, however, must be influenced by the criticality of the particular test cases.

Additionally, the quantity of test cases performed should not influence confidence in the system. A large number of test cases that exercise a relatively small portion of the test domain contribute minimally to user confidence. At issue is not the quantity, but coverage of the test cases. A good representative sampling of the input domain is necessary to build confidence in the system.

For conventional software, the sequence of execution corresponds to the high level code. In an ES, the exact sequence of execution may be difficult to determine by inspection. This becomes an issue since it adds to the difficulty of determining test coverage for verification. However, the sequence of execution is known after execution, since ESs can justify their reasoning by showing the chain of logic used to reach a conclusion.

5.5.5.4 Expert System Black Box Techniques.

Black box techniques examine whether or not the solution of a particular problem fits, or solves, that problem. They are independent of any implementation language or design.

5.5.5.4.1 Knowledge Acquisition Checking.

The earliest that errors in knowledge are introduced into the development process is in the acquisition of knowledge from the expert. Errors at this stage may be kept to a minimum by testing the expert knowledge for completeness. This can be accomplished by checking for the following (French and Hamilton 1992):

- Tasks which are defined but have no associated goal
- Goals that are defined with no associated task
- Undefined terms
- Conditions for which no goal or task is defined

5.5.5.4.2 Knowledge Consistency.

Consistency of the supplied knowledge is also the focus of V&V techniques. This involves checking for contradictory or redundant statements.

5.5.5.4.3 Knowledge Representation.

Representing knowledge in a form easily understood by both the expert and the knowledge engineer is essential to the verification process. Some techniques that can be used to simplify the representation of knowledge include state tables, concept maps, and task time lines (French and Hamilton 1992).

5.5.5.5 Formal Verification of Expert Systems.

Formal methods, a technique designed for verification of conventional software, can also be used in verification of ESs. This technique, described as an alternative technique in RTCA/DO-178B, can be used to satisfy airborne software certification requirements. According to Rushby and Whitehurst (1989), this technique "can, in principle, guarantee the absence of faults." Its application involves demonstrating consistency between two different descriptions of a program. This can be done using two different levels of program specifications, or by using the program itself along with the program specification.

Consistency can be demonstrated by the working out of a formal mathematical proof, when the two parts are treated as formal mathematical texts. One way this technique can be applied is by using a "stacked" application of verification steps. Conventional software breaks down into a number of steps during the development process. At each step it is possible to represent each specification level formally and to demonstrate that each successive level demonstrates consistency with the prior level.

Correctness of the executable code is guaranteed provided:

- The proofs are error free.

- The real requirements are represented accurately in the requirements statement.

- An appropriate concept of consistency is employed for the intended application.

- The program execution behavior is an accurate reflection of the program code during its verification.

ESs do not seem to be the ideal choice for application of formal verification techniques for several reasons:

- The requirements are often vague. Clarification will need to be performed on vague requirements before they can be represented as formal mathematical statements.

- The developmental process for ESs does not generally mirror that of conventional software, for which formal methods were designed.

- Formal methods were designed for application to algorithmic software. ESs use search techniques applied to heuristic information.

- Knowledge representation methods for ESs were not designed to suit the needs of formal verification.

Rushby and Whitehurst suggest a method that may be used to arrive at a solution for use of formal methods with ESs. They suggest first separating the conventional aspects from the AI aspects of the system. This is done by identifying and separating the competency (knowledge and human skill) requirements from the service (conventional software) requirements. Formal verification techniques may be applied to the service requirements since they are similar to conventional software.

Once the competency requirements are identified, they are further divided into "desired" and "minimum" requirements. Rushby and Whitehurst indicate that it would be difficult to model explicitly the desired competency requirements, but that the minimum competency requirements may be defined precisely. Further, they feel that these requirements "may be able to capture certain desired safety properties for AI software and that such requirements often can be cast as formal requirements specifications." (Rushby and Whitehurst 1989).

The use of formal methods is a promising technique, but requires further research. The formal methods technique can be time consuming and difficult, and, therefore, expensive to apply. Since this is so, formal methods are typically reserved for critical applications, where reliability and safety issues are primary considerations. This technology would benefit from guidelines or standards for expressing knowledge that needs to be cast into the formal verification mold. In addition, the development and use of suitable tools could assist ES implementers in producing systems with greater reliability.

5.5.5.6 Other Verification Techniques.

Other black box methods used in V&V of ESs include expert review, minimum competency testing, and disaster testing. Expert review is necessary for systems requiring V&V. By definition, the expert already has an understanding of the problem to be solved and may be able to work out solutions to test cases. Having the implementation in a form easily understood by the expert is essential to this phase of the review.

A minimum competency testing of an ES involves testing the system in much the same way as humans are tested for proficiency. A number of different scenarios may be generated for testing the response of the ES.

Systems are often tested for scenarios that may rarely, if ever, occur. Typical operation of the system under normal conditions often is treated with less rigor than some of the remotely possible conditions. These remotely possible conditions, however, may have serious consequences when they do occur. Testing for conditions such as these is referred to as disaster testing. This type of testing is also necessary for testing completeness of the system.

5.5.5.6.1 Automating Knowledge Base Verification.

Some tools are designed to assist the developer in automating the knowledge acquisition process. Others can keep the knowledge engineer in charge of the knowledge transfer process. As ESs become more commonplace and take on critical tasks, the need for tools to assist in the validation process increases. Methods to automate the validation process can provide many benefits. Tools can shorten the verification process, eliminate bias, and catch problems that may otherwise be overlooked. Following are the types of problems that may be found using automated tools:

- Syntactic errors
- Unused rules, facts, and questions
- Incorrectly used legal values
- Redundant constructs
- Rules that use illegal values, incorrect external declarations or references, and multiple methods for obtaining expression values

Jafar and Bahill (1993) describe a tool called Validator. This tool provides an interactive environment that is used to identify certain types of errors that may exist in knowledge-based systems. Syntax errors, misspelled words, values out of range, and unused values and rules are some of the errors detected by Validator. It has been exercised on a number of ES knowledge bases produced by university students, in addition to demonstration system knowledge bases from commercial software houses.

Table 5.5-4 shows the results of running Validator on seven ESs that were either masters projects or precommercial system releases.

The user must be aware of certain limitations of the tools. For instance, Validator does not identify errors such as the following:

- Circular and recursive rules
- Compound structures such as frames or object data types and their anomalies
- Conflicting rules
- Redundancies inside of rules

TABLE 5.5-4. ERRORS DETECTED BY VALIDATOR
(Jafar and Bahill 1993)

Expert System	Size (Kbytes)	Unused Rules	Anomalies	Unused Questions	Multiple Questions	Multiple Methods	Referential Integrity
Advice	82	3		9		35	1
Chromie	100	1					
Cogito	126	5	25		6	7	
Fundeye	60						
Helper	50			1			14
SRPRES	120	2	2				1
Stutter	81	12		5	1		8

Regardless of the tool developers use, tool limitations should be identified and other methods used to locate these errors.

5.5.6 Supportability Issues for Expert Systems.

In a systems engineering environment, supportability issues are not considered after a product is fielded. They are considered at the requirements level and influence the specification, design, and testing.

Following are several techniques used to enhance supportability of ESs (Carlton 1987):

- Keeping the inference engine separate from the knowledge base – Allows the knowledge base to be maintained without inference engine changes, allowing ease of use and flexibility.

- Representing knowledge uniformly – The higher the number of structures used to represent knowledge, the greater will be the effort when updates are required. It could be difficult to maintain a single mechanism since the problem type may dictate the most efficient representation.

- Avoiding complexity in the inference engine – For cases where the inference engine is developed by the user, a well-designed and simplified engine facilitates the generation of explanations.

- Utilizing redundancy – When developing the knowledge base, the use of multiple knowledge sources is recommended so that confidence in system performance is increased.

Another supportability issue focuses on the development environment used. If an ES shell is used, development can be faster and more reliable. Additionally, maintenance can be performed on this type of system by nonprogrammers. If an HOL is used, the developer has greater flexibility with knowledge representation and inferencing schemes. Additionally, if an HOL is used, other considerations should be weighted, such as ease of use, expressiveness, and flexibility of the HOL, since these factors influence supportability.

5.5.7 Configuration Management Issues for Expert Systems.

CM is a process or discipline that manages and controls the modification of hardware and software components of a system. It defines functional and physical component characteristics and records and reports changes in the status of these components. It should be applied rigorously to large-scale software and hardware development efforts. It has classically been applied to procedural software.

Knowledge can be represented both procedurally and declaratively. Procedural solutions are derived after a sequence of instructions is executed. An inference engine is an example of procedural knowledge. Conventional programs are totally procedural, while knowledge bases may contain a mix of procedural and declarative knowledge. Declarative knowledge consists of facts asserted to be true. An inference scheme is also associated with these facts (Carlton 1987).

CM techniques applied to procedural software are understood and can be applied easily. Since the inference engine is procedural, CM for it is well defined. Changes to the knowledge base have no influence on the inference engine. According to Carlton (1987), conventional CM for the knowledge base presents a problem. He states:

> While conventional configuration management procedures can be applied to the knowledge base, this will put unnecessary constraints on the flexibility of the knowledge-based system by limiting the ability of the user to make changes to the knowledge base to suit local conditions.

Changes to the knowledge base may become necessary when the ES operating environment changes. Local conditions are considered when information contained in the knowledge base is tailored to suit each operating environment. In this case, rapid changes to the knowledge base are required and the conventional CM process will be burdensome.

A solution to this problem is suggested. If the knowledge-based system can track its own configuration, much of the CM burden can be alleviated. This can be accomplished by having the system track changes and note them in a configuration data file. This file can be transmitted to the CM facility periodically. This file is then subject to central control and analysis (Carlton 1987).

5.5.8 Reliability Requirements.

For certificating systems that will be used for flight critical functions, designers can take one of two approaches. The first approach, "safe-life," means the component is designed to keep its strength and integrity throughout its life. The second method, "fail-safe," means safety is ensured by having a redundant or back-up part that will work if the first component fails (Improving Aircraft Safety 1980).

The fail-safe approach has been adopted for digital avionic systems. Because of the risk of applying complex technology to critical or essential functions, avionic systems used to support either category usually are implemented with redundancy, either similar or dissimilar.

If enough redundancy ensures that the probability of an unsafe event occurring is acceptable (not greater than 1×10^{-9} for critical functions), a digital system may be used to replace the previous analog or electromechanical system.

This type of approach worked well for systems that were not very complex. Due to rapid growth in digital technologies in recent years, assessing reliability of current digital systems is a growing concern (Keller 1992). Reliability figures for software are essentially figures of merit reached after the software is examined using various techniques or metrics.

Metrics can be valuable sources of indicators for both development cost and failure intensity. Two common metrics applied to conventional software include complexity and size. The assumption is that larger programs and more complex programs will contain more errors and, therefore, be less reliable. A major difficulty is knowing which metrics to use and how to interpret the results (Rushby 1988). Measuring reliability is further complicated due to the differences between knowledge-based and conventional software. Further information on software metrics can be found in DOT/FAA/CT-91/1, "Software Quality Metrics."

Since ES technology is still rather young, there is a lack of experimental data for these systems. It also may be risky to assume that the relationships observed for conventional software will apply in the same way to ESs. Concerning this, Rushby (1988) states:

> Empirical data will need to be gathered in order to suggest and validate useful relationships among metrics for AI software. It is unlikely that simple counts of the number of statements or rules in an AI system's knowledge base will yield useful information regarding its competence.

5.5.9 Testability Issues for Expert Systems.

5.5.9.1 Testing Techniques.

With conventional software, coverage techniques exist for testing. They include branch, path, and condition coverage. Critical paths can be examined and traced through the software. The same is not true for knowledge-based software, since no critical paths exist to trace.

There are a large number of possible combinations and permutations of knowledge. This necessitates a change of testing methods for knowledge bases. Additionally, the number of test cases available to test the operation and performance of the ES will influence the testing coverage and the user's confidence in the system.

5.5.9.2 Validity of Input Data.

For conventional software, input data are tested for validity by verifying that they are within acceptable bounds as defined for the system. Methods for checking the validity of largely symbolic data need to be examined and developed for knowledge-based systems.

5.5.9.3 Testing Level for Changes to the Knowledge Base.

A change made to the knowledge base of an ES used in a critical system requires testing. The level of testing required is unclear. Working with partitioned knowledge bases may mitigate some of the difficulties encountered when testing changes (Carlton 1987).

6. HUMAN FACTORS AND ARTIFICIAL INTELLIGENCE.

Human factors concerns cover all aspects of technology, from design through implementation. To field automated intelligent systems and pilot assistants that add to the safety of flight, human factors technology should be applied to designs from the beginning. This will ensure that potential problems are identified and corrected before they have significant impact on system development.

6.1 EXPERT SYSTEM DEVELOPMENT STAGES.

There are four critical stages of development for ESs and their human factors-related concerns, as identified by Wickens (1992):

- Knowledge Acquisition – For complex ESs, the process of extracting knowledge from experts, and implementing that knowledge in the form of a rule base, can be a considerable challenge. Information supplied by the expert is subject to bias for which "debiasing" techniques may need to be applied. Additionally, not all expert knowledge can be verbalized. Methods other than verbalization may need to be explored.

- Knowledge Representation – How knowledge is organized is a concern. Is it understandable and can it be retrieved easily by users? Also, consideration is necessary for tailoring the stored knowledge to the capability level of the user.

- Knowledge Utilization – The design of the computer interface must consider sound human factors design techniques. This includes displays and controls, as well as the vocabulary used to communicate with the user. Another factor that needs to be addressed is how the confidence level of the ES's offered advice (which is a variable) is communicated to the user.

- Knowledge Maintenance – As is typical with any computer system, its data occasionally require revision. Knowledge bases of ESs also require maintenance, either with new information being added or older knowledge being modified. How this is done, and when it is done, are concerns that certification specialists need to address.

6.2 NATIONAL PLAN FOR AVIATION HUMAN FACTORS.

A valuable resource for identifying human factors issues and related safety concerns is the FAA's National Plan for Aviation Human Factors. This Plan represents the first step of an intensive effort to coordinate human factors-related research in aviation among government agencies, such as the FAA, NASA, and DOD.

The purpose of the National Plan includes identifying technical efforts necessary to address human performance issues in aviation, acquiring the necessary resources for funding these efforts, allocating resources and coordinating programs, communicating research needs, and promoting the means by which human factors knowledge is transferred to Government and industry. Some of the Plan's objectives include the following:

- Encouraging the development of principles of "human-centered" automation and the design of advanced technology that will capitalize upon the relative strengths of humans and machines.

- Improving aviation system monitoring capability, with an emphasis upon human performance factors.

- Encouraging the improvement of basic scientific knowledge and facilitating understanding of the factors that significantly influence human performance in aviation.

- Developing better techniques for assessing human performance in the aviation system.

- Developing human factors-oriented validation and certification standards for aviation system hardware and personnel that will enhance both safety and efficiency.

6.3 NATIONAL PLAN AND ARTIFICIAL INTELLIGENCE.

In the Plan, a number of issues are raised that focus on safety concerns for intelligent cockpit devices. One such concern is how pilots will react to probabilistic AI. The Plan states:

> Automated flight deck systems will soon incorporate artificial intelligence with the capability to give pilots information or advice about complex matters with an associated probability that it is correct. Before such probabilistic AI systems come into extensive use in advanced technology aircraft, we need to know more about how crews will react to them, how to design them for optimum use, and how to train crews to interpret and use the probabilistic information.

In addition, other concerns, such as those listed below, are identified and expressed in the Plan:

- Intelligent automated systems should not be able to move the aircraft out of the safe operating envelope.

- The test and validation methodology for intelligent automated systems is still in its infancy and requires research and definition.

- The reaction of the flight crew to the type of information presented by an AI system requires investigation.

6.4 KEEPING THE HUMAN AS A FACTOR.

One of the human factors problems identified for automated systems is the user's overconfidence in the decision making process. According to Wickens (1993), one of the methods proposed to:

> deal with bias, one that is emerging in the commercial flight deck and is already used in the military flight deck... [is the use of expert system technology]. Expert systems can, at least according to some scientists, replace some of the pilot decision making necessary in the cockpit, or at least can recommend judgements to the pilot...

Automated decision aiding has been studied for its ability to reduce or eliminate bias from decision making. There is some concern that these aids may serve to worsen, and not improve, decision performance (Wiener and Nagel 1988). In many of the cases where poor performance was noted, the human was used to gather data and implement decisions of the computer. Figure 6.4-1 depicts the interaction using this method. In some cases, the combination led to worse performance than either the machine or human acting alone.

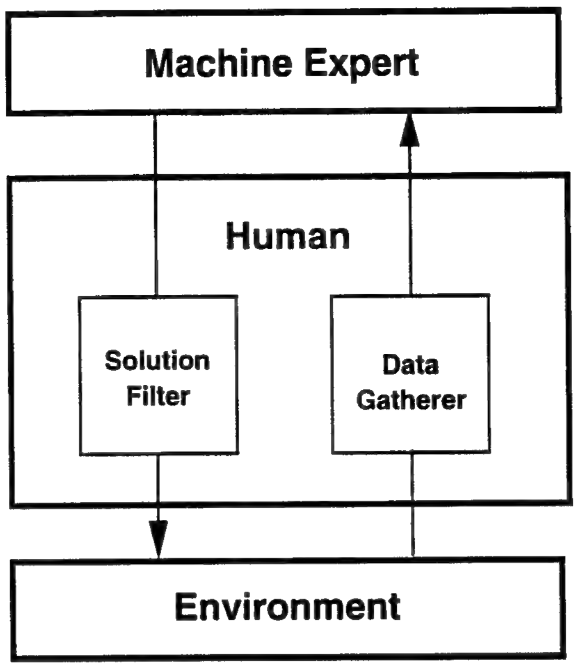


FIGURE 6.4-1. POOR HUMAN PERFORMANCE GENERATOR
(Wiener and Nagel 1988)

A method of improving the overall performance and interaction is shown in figure 6.4-2. This method takes into account the capabilities and strengths of the human and the machine. All of the possible solutions are generated by the machine. A more active role for the human is maintained using this method, and, hence, the human remains in the loop in a key role; not merely as a physical actuator for control input devices.

Keeping the pilot in the loop is a concern, whether due to use of AI or due to use of any cockpit automation system. Reliance on yet another system may lead to further complacency and inattentiveness. Failure of an automated system creates increased crew workload, often at a time when this additional workload is unwelcome.

Any system, whether analog or digital, is prone to failure. A number of factors, including system complexity and unforeseen circumstances, prevent avionic systems from operating error free over their life cycle. The unintentional changing of a single bit of a digital system, whether by a software or hardware design error or externally induced, can produce significant undesirable consequences. While it may be possible for the system to monitor itself and flag most of these errors, not all problems are identified and announced to the system operator. Other systems may be used to monitor system health and reduce the probability of an undetected failure. Therefore, the role of the human as a system monitor should remain, regardless of the level of automation.

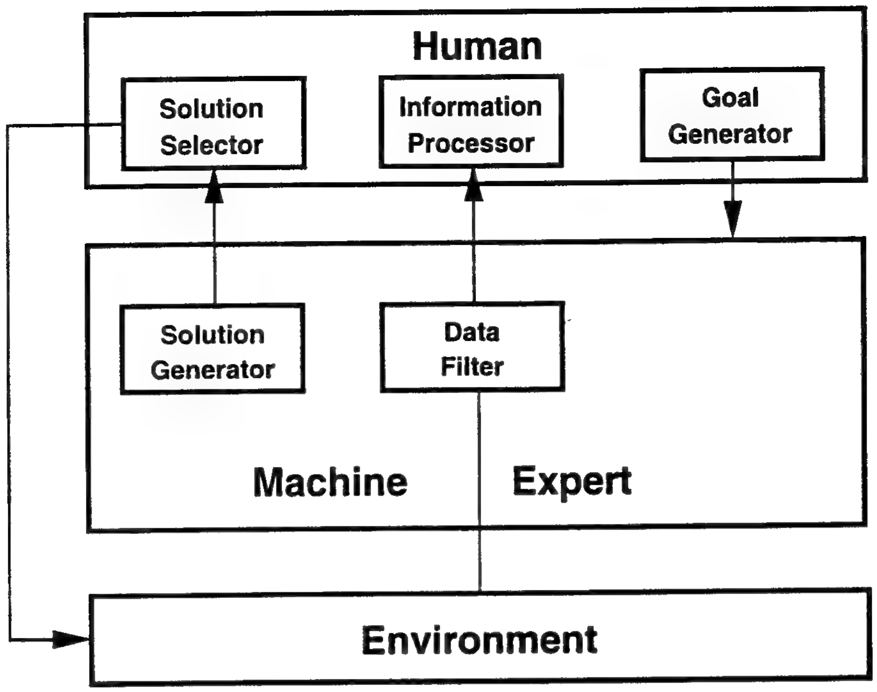


FIGURE 6.4-2. IMPROVED HUMAN PERFORMANCE GENERATOR
(Wiener and Nagel 1988)

6.5 AUTOMATION AND OTHER HUMAN FACTORS CONCERNS.

Other factors need to be examined, in addition to the problem of keeping the pilot in the loop. Human factors concerns over automation systems reflect some of the same concerns presented by AI systems. Some of these concerns are identified by Gabriel (1993) and consist of the following:

- Loss of Situation Awareness – Monitoring an automated system that is reliable is a boring task. The crew becomes distracted easily and diverts attention elsewhere. When this happens, awareness of the current situation is diminished.

- Loss of Proficiency – Competence in a skill requires practice. How to retain skills that are replaced or diminished by an automated system is a human factors issue.

- Reduced Job Satisfaction – Reduced job satisfaction can occur since the crew may no longer feel that their skills are important.

- Overconfidence in the System – Overconfidence in the system is a negative effect of automation. If the system malfunctions, the crew may not be prepared to handle the consequences.

- Intimidation by Automation and/or Complacency – A crew lacking experience or being overconfident may be reluctant to take control when the automated system does not perform well. In addition, an automated system may be designed in a way that does not communicate everything that the crew should know. This reluctance and uncertainty may reduce the ability of the crew to fulfill its responsibility.

- Increased Training Requirements – System complexity requires increased training. Part of the required training is to handle failure modes. Reprogramming can also be complex. Maintaining proficiency is necessary and, therefore, increased training is required.

- Inability of the Crew to Exercise Authority – This results when the crew is further removed from the decision making process by such factors as intimidation and the envelope protection features of the system.

- Design Induced Error – One of the goals of automation is to reduce human error. A concern is that while the frequency of errors is decreased, the consequence of the errors that do occur is much greater due to the control exerted by the automated system. Concerning this, Nigel (in Wiener and Nigel 1988) states:

  > As aircraft become more highly automated, and the manual control skills of the pilot are displaced by aircraft systems (autopilots) which perform the general guidance and control functions associated with management of the flight path, discrete actions... seem to be assuming increasing importance in the causation of incidents and accidents.

7. CONCLUSION.

AI technology can provide users with many benefits in a number of applications. However, there are limitations as to what can be performed with an AI-based system. It will never be able to replace genuine intelligence. AI deals with automating procedures that are already known. If little is known about a procedure, it is better to research the procedure than to attempt development based on vague specifications.

With currently available technology, ESs perform better as assistants or advisors than as primary decision makers. Humans will always have the advantage over ESs in that they have the power of human sensory pattern recognition and flexibility. AI systems will not be capable of replacing humans in most applications and will be used mainly to augment the capabilities of the user.

Another issue that needs to be addressed is the proper application for an AI-based system. AI technology is not efficient at solving all types of problems. Intense number manipulation and linear problems would be better performed using conventional methods. There are certain types of problems that naturally lend themselves to being solved using AI techniques.

Human experts often are in short supply. ESs are useful because the knowledge of experts can be incorporated into knowledge bases, allowing users who are not experts to benefit from the expert's knowledge. NN systems are helpful for applications such as pattern recognition. Fuzzy logic systems are useful for systems that normally require human intuition, are difficult to control with conventional techniques, or are difficult to model.

While many benefits can be gained from the use of AI-based technology, a number of issues need to be addressed before such systems are fielded in critical applications of airborne avionics. This section summarizes and highlights some key issues concerning certification of AI-based technology.

7.1 LEARNING SYSTEMS.

One of the certification concerns mentioned in the National Plan for Aviation Human Factors is that airborne software not be allowed to remove the aircraft from the safe operating envelope. The capability of software to maintain an aircraft within the design envelope can be demonstrated for conventional, deterministic software. However, this would be difficult to demonstrate for AI-based systems with learning capability. Therefore, learning systems are unlikely to appear in civil aircraft (Naser 1991).

7.2 SOFTWARE LEVEL DETERMINATION.

While ESs have been proposed and may be developed for military aircraft for the purpose of intervening and assuming control in life-threatening situations, it is unlikely that this would happen in the near future for civil aircraft. It is more likely that ESs introduced in the near future for civil use will perform monitoring and pilot assistant functions. Certification specialists most likely will not require these systems to undergo the rigorous inspections and tests to which critical systems are subjected. One may argue that since the system actually controls nothing, it is not able to affect the performance and safety of the aircraft. On the other hand, receiving incorrect advice from a system that is normally relied upon by the captain and first officer can have dire consequences. Certification authorities will need to address these issues early in the development process, since determination of the software criticality level of a system affects inspection and documentation intensity and, therefore, implementation cost.

7.3 ENHANCING SAFETY.

An objection may be raised that we are just adding another layer of uncertainty on top of a system that is already plagued with unknown failures. Digital systems have unique failure modes compared to analog systems. Failures that are detected in aircraft operation at 40,000 feet are sometimes not reproducible on the test bench. Some digital systems problems magically disappear when the breaker is cycled off and on by the flight crew. Additionally, failures of digital systems are not always detected in a timely fashion. With this level of uncertainty already existing, would the addition of another level of uncertainty further degrade safe flying?

A similar objection can be raised when multiple access data buses or high-speed parallel backplanes are implemented for critical systems. Technologies formerly unknown in avionics are continually being designed and implemented and certification specialists address these issues as they arise. At issue is how AI and ES technology are perceived. If confidence in the technology is lacking, then its application to any application is viewed as a degradation. If it is perceived as a sufficiently mature technology that excels in certain applications (as it does), then its application in cockpit systems is seen as a benefit.

7.4 HARDWARE AND SOFTWARE FAILURES.

There are numerous ways to express the same functionality using conventional software for system design. As tools automate more of the development process, design is performed using higher levels of representation.

One architectural design technique, mentioned in RTCA/DO-178B, for limiting the impact of software errors is the use of redundant, dissimilar software. The degree of dissimilarity and the advantages gained by utilizing this technique are minimized by the higher levels of representation available to designers. For ESs, this is also the case, since one of the attractions of this technology is that knowledge can be coded at a high level. Redundancy implies that the expert's rules would have to be different for accomplishing the same ends. Knowledge base partitioning and system monitoring techniques, along with extensive, detailed V&V, may be the solutions needed for ESs designed to perform critical functions.

The effect of hardware failures for AI-based systems also needs to be examined. What effect will hardware failures have on the conclusions reached by an ES? In the case of a pilot-aiding system, will failures be detectable only by the bad advice given by the system? Hardware error detection schemes will be critical to ensure system reliability, as they are with any automated system.

7.5 GROUND-BASED AVIONICS?.

There is a related issue when considering the certification of AI-based airborne software. Some proposed pilot aid systems rely upon the availability of data link to receive company-related information required by the system. Data link is a two-way digital communication system. Using data link, one could transmit an aircraft's complete status, environmental data, and other necessary information to a ground-based facility on a communication channel. If that channel has sufficient bandwidth for a timely response, the exact AI-based program could be executed at the ground-based facility with the results transmitted back to the aircraft.

Such a system would not be subject to the certification requirements for airborne software, yet the only significant difference would be the physical location of the CPU that executes the AI-based software. While such a scenario may be impractical at the current time, certification specialists should consider the impacts of relying on this technology for airborne support, wherever the data are generated.

7.6 VERIFICATION AND VALIDATION ISSUES.

One of the primary activities that requires addressing by developers and certification specialists concerns V&V methods for AI systems. Numerous V&V techniques that have been successfully applied to ESs have been identified in this report. Identifying a set of tests that can be demonstrated to satisfy safety requirements will be a challenge.

All parts of the ES, such as the database, knowledge base, inference engine, interfaces, and custom software need to be subjected to suitable inspection. Knowledge bases need to be complete. Missing rules will produce errors. For complex ESs, demonstrating that the knowledge base is complete may be difficult, since the expert may not even be aware of what is done to solve a problem. Also, it is impossible to guarantee that an ES contains all of the expert's knowledge.

The knowledge base also needs to be error free. Many of the techniques used with conventional software may be used to demonstrate that the system will not produce incorrect results. Also crucial to the knowledge base is maintaining data integrity. Maintaining an accurate, up-to-date, database is essential for product integrity throughout the product life cycle. Knowledge acquisition, representation, and data transformation techniques should be tested.

As with conventional software, key to the certification process for ESs will be the faithful application of a thorough V&V process, applied uniformly and starting with the product specification. Life cycle factors such as supportability, maintainability, and CM need to be considered and tailored for ESs as necessary.

7.7 TOOL QUALIFICATION.

Tools at all levels of the development process can assist in achieving cost, safety, and other goals. Tools enhance the development process by reducing development time and assisting in the elimination of human design error. Tools to automate knowledge base verification could provide substantial benefits when addressing knowledge base integrity issues. Standardization of knowledge base representation may be necessary to implement the automation process. Also, tool qualification issues require addressing since a new set of tools focusing on this field will require validation.

7.8 DESIGN GUIDANCE.

ESs should be scrutinized using conventional V&V techniques where possible. RTCA/DO-178B needs to be examined to identify details of the processes that need to be modified to accommodate differences in AI-based and conventional development. Further effort is needed to develop relevant design guidance for AI-based systems.

7.9 HUMAN FACTORS ISSUES FOR EXPERT SYSTEMS.

Developing human-centered automation and designing advanced technology that will capitalize on the relative strengths of humans and machines are key to the success and usefulness of AI. Also, applying certification criteria to advanced technology systems, as part of the development process, will provide significant advantages for the testing phase.

The use of AI on the flightdeck brings up many issues relating to automation that need to be dealt with. Human factors specialists will need to address areas such as possible loss of situation awareness and proficiency, in addition to overconfidence in the system or intimidation by the system. The overall significance of incorporating probabilistic systems in the cockpit should be addressed as well.

Wickens (1993) states that too little is known about the way pilots make decisions and, therefore, ESs should not be entrusted with decision recommendations. While this is certainly a concern and has been the focus of much research for the PA and other studies, the many benefits of using AI-based technology beg to be investigated. At issue is not the complete understanding of the human thought process, but using a logical, consistent, unbiased decision tool (inference engine) to make conclusions based on available data. Simple systems, such as diverters and diagnostic aids based on ESs present logical approaches to solving complex problems in an accurate and timely fashion.

AI-based systems for commercial aircraft are being researched and tested. These systems can offer distinct advantages over conventional software systems. Ultimately, such systems can offer economic advantages and contribute to flight safety. AI-based pilot aiding systems have the potential to contribute meaningfully to aviation. A solution that matches the technology with the problem to be solved, using a systems engineering approach, will be a solid start for developers of AI-based avionics.

BIBLIOGRAPHY

Adeli, Hojjat, Knowledge Engineering Volume 1, Fundamentals, McGraw-Hill Publishing Company, New York, NY, 1990.

Anderson, Bruce A., et al., "Knowledge Engineering for a Flight Management Expert System," Proceedings of the IEEE 1985 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1985.

Anderson, James A., "Neural-Network Learning and Mark Twain's Cat," IEEE Communications, Volume 30, No. 9, September 1992.

Antonisse, James H. and Karl S. Keller, "Dynamic Evaluation of Imprecisely Specified Knowledge," IEEE/AIAA Digital Avionics Systems Conference Proceedings, IEEE Service Center, Piscataway, NJ, 1986.

Aptronix[1], Inc., NEWS11.TXT, San Jose, CA, July 1992.

Aptronix[2], Inc., What is Fide?, San Jose, CA, July 1992.

Araya, Agustin and Sanjay Mittal, "Compiling Design Plans from Descriptions of Artifacts and Problem Solving Heuristics," Proceedings of the 10th International Joint Conference on Artificial Intelligence, Milan, Italy, August 23–28, 1987.

Aviation Week & Space Technology, "Software Developed for Using Neural Network Techniques in Problem Solving," June 17, 1991.

Avionics Update, "Army Awards Contract for "Intelligent" Helicopter Associate," Avionics, September 1993.

Ballard, Dan and Dave Nielsen, "A Real-Time Knowledge Processing Executive for Army Rotorcraft Applications," IEEE/AIAA 11th Digital Avionics Systems Conference Proceedings, IEEE Service Center, Piscataway, NJ, 1992.

Ballard, Dan and Lisa Owsley, "Artificial Intelligence in the Helicopter Cockpit of the Future," IEEE/AIAA 10th Digital Avionics Systems Conference Proceedings, IEEE Service Center, Piscataway, NJ, 1991.

Barber, Richard and George Imiah, "Delivering the Goods with Lisp," Communications of the ACM, Volume 34, No. 9, September 1991.

Bartee, Thomas C., ed., Expert Systems and Artificial Intelligence: Applications and Management, Howard W. Sams and Company, Indianapolis, IN, 1988.

Becker, Lee, et al., "Translating Expert System Rules into Ada Code with Verification and Validation," NASA Contractor Report 187505, NASA Langley Research Center, Hampton, VA, June 1991.

Berardi, L., et al., "System Architectures," AGARD Conference Proceedings 417: The Design, Development and Testing of Complex Avionics Systems, Advisory Group for Aerospace Research and Development, Neuilly Sur Seine, France, December 1987.

Berning, Sandra, Douglas P. Glasson, and James A. Guffey, "Adaptive Tactical Navigation Concepts," Proceedings of the IEEE 1986 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1986.

Berning, Sandra, Douglas P. Glasson, and Gary A. Matchett, "Functionality and Architectures for an Adaptive Tactical Navigation System," Proceedings of the IEEE 1987 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1987.

Berning, Sandra, Douglas P. Glasson, and James A. Guffey, "Adaptive Tactical Navigation Program," AGARD Conference Proceedings 499: Machine Intelligence for Aerospace Electronic Systems, Advisory Group for Aerospace Research and Development, Neuilly Sur Seine, France, May 1991.

Berning, Sandra, Douglas P. Glasson, and Jean-Michael L. Pomarede, "Knowledge Engineering for the Adaptive Tactical Navigator," Proceedings of the IEEE 1988 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1988.

Bezdek, James C., Analysis of Fuzzy Information, CRC Press, Boca Raton, FL, 1987.

Bezdek, James C., "Computing with Uncertainty," IEEE Communications, Volume 30, No. 9, September 1992.

Bezdek, James C. and Pal K. Sankar, Fuzzy Models for Pattern Recognition, IEEE Computer Society Press, Los Alamitos, CA, 1991.

Biema, Michael, "Parallelism in Lisp," Proceedings of the 10th International Joint Conference on Artificial Intelligence, Milan, Italy, August 23–28, 1987.

Bittermann, Vincent, et al., "FINDER, A System Providing Complex Decision Support for Commercial Transport Replanning Operations," IEEE Aerospace and Electronic Systems, Volume 9, No. 3, March, 1994.

Bittermann, Vincent, et al., "FINDER: Flight-Plan Interactive Negotiation and Decision Aiding System for Enroute Rerouting," 12th International Conference on Artificial Intelligence, Expert System, Natural Language, Avignon, France, June 1–6, 1992.

Boehm, Barry W., "A Spiral Model of Software Development and Enhancement," Uma G. Gupta ed., Validating and Verifying Knowledge-Based Systems, IEEE Computer Society Press, Los Alamitos, CA, 1991.

Bouchard, Philippe O., Keynote Address, AGARD Conference Proceedings 499: Machine Intelligence for Aerospace Electronic Systems, Advisory Group for Aerospace Research and Development, Neuilly Sur Seine, France, September 1991.

Bowyer, M. R. and S. A. Cross, "Parallel Knowledge Based Systems Architectures for In-Flight Mission Management," AGARD Conference Proceeding 504: Air Vehicle Mission Control and Management, Advisory Group for Aerospace Research and Development, Neuilly Sur Seine, France, October 1991.

Boys, Randy and Katherine Palko, "Automation and Dynamic Allocation: Engineering Issues and Approaches," Proceedings of the IEEE 1988 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1988.

Breuker, Joost, et al., "A Shell for Intelligent Help Systems," Proceedings of the 10th International Joint Conference on Artificial Intelligence, Milan, Italy, August 23–28, 1987.

Briot, Jean-Pierre and Pierre Cointe, "A Uniform Model for Object-Oriented Languages Using the Class Abstraction," Proceedings of the 10th International Joint Conference on Artificial Intelligence, Milan, Italy, August 23–28, 1987.

Brower, Ronald W., Larry E. French, and Richard W. Linderman, "LISP Garbage Collection Using Content-Addressable Memory," Proceedings of the IEEE 1987 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1987.

Brown, David and John M. Carson, "Embedded Expert Systems for Avionics Applications," IEEE/AIAA Digital Avionics Systems Conference Proceedings, IEEE Service Center, Piscataway, NJ, 1986.

Brubaker[1], David L., "Fuzzy-logic Basics: Intuitive Rules Replace Complex Math," EDN, Volume 37, No. 13, June 18, 1992.

Brubaker[2], David L., "Fuzzy-logic System Solves Control Problem," EDN, Volume 37, No. 13, June 18, 1992.

Brubaker, David L., "Everything You Always Wanted to Know About Fuzzy Logic," EDN, March 31, 1993.

Brule, James F., "Fuzzy Systems – A Tutorial," 1985.

Butler, G. F. and M. J. Corbin, "FLEX: FORTRAN Library for Expert Systems," RAE Working Paper, MM 273/88, December 1988.

Carlton, Kyp A., "Artificial Intelligence Supportability," Proceedings of the IEEE 1987 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1987.

Carpenter, Gail A. and Stephen Grossberg, "A Self-Organizing Neural Network for Supervised Learning, Recognition, and Prediction," IEEE Communications, Volume 30, No. 9, September 1992.

Chappell, Alan R., "Knowledge-Based Reasoning in the Paladin Tactical Decision Generation System," IEEE/AIAA 11th Digital Avionics Systems Conference Proceedings, IEEE Service Center, Piscataway, NJ, 1992.

Charniak, Eugene and Drew McDermott, Introduction to Artificial Intelligence, Addison-Wesley Publishing Company, Reading, MA, 1985.

Chen, David C., "An Expert Planner for the Dynamic Flight Environment," Proceedings of the IEEE 1985 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1985.

Clarkson, Douglas, "Living Computers," Electronics World and Wireless World, March 1990.

Clocksin, W. F. and C. F. Mellish, Programming in Prolog, Springer-Verlag Berlin, Heidelberg, Germany, 1984.

Coats, Pamela K., "Why Expert Systems Fail," Uma Gupta ed., Validating and Verifying Knowledge-Based Systems, IEEE Computer Society Press, Los Alamitos, CA, 1991.

Cochran, Keith G., "Artificial Intelligence Techniques Applied to Vehicle Management System Diagnostics," IEEE/AIAA 10th Digital Avionics Systems Conference Proceedings, IEEE Service Center, Piscataway, NJ, 1991.

Cohen, Paul R., Heuristic Reasoning About Uncertainty: An Artificial Intelligence Approach, Pittman Advanced Publishing Program, Boston, MA, 1985.

Coleman, Lisa A., "Lockheed First to Test Neural Net Chip," Military and Aerospace Electronics, Volume 4, No. 3, March 15, 1993.

Conner, Doug, "Designing a Fuzzy Logic Control System," EDN, Volume 38, No. 7, March 1993.

Cox, Earl, "Fuzzy Fundamentals," IEEE Spectrum, Volume 29, No. 10, October 1992.

Cox, Earl, "Adaptive Fuzzy Systems," IEEE Spectrum, Volume 30, No. 2, February 1993.

Culbert, Chris, Gary Riley, and Robert T. Savely, "Approaches to the Verification of Rule-Based Expert Systems," Uma G. Gupta ed., Validating and Verifying Knowledge-Based Systems, IEEE Computer Society Press, Los Alamitos, CA, 1991.

Curran, Jim, Trends in Advanced Avionics, Iowa State University Press, 1992.

Day, P. O. and M. K. Hook, "An AI-Based Fault Diagnosis Aid for Complex Electronic Systems," IEEE/AIAA 8th Digital Avionics Systems Conference Proceedings, IEEE Service Center, Piscataway, NJ, 1988.

Daysh, Colin, et al., "A NASA/RAE Cooperation in the Development of a Real-Time Knowledge Based Autopilot," AGARD Conference Proceedings 499: Machine Intelligence for Aerospace Electronic

Systems, Advisory Group for Aerospace Research and Development, Neuilly Sur Seine, France, May 1991.

Decision Aiding System for Commercial Aircraft, Sextant Avionique Flight Control Systems Division, Briefing to FAA Technical Center, Atlantic City International Airport, NJ, February 1993.

Deeb, Joyce M. and Andrew G. Philpot, "Data Management in Large-Scale AI Systems," Proceedings of the IEEE 1988 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1988.

Delgrande, James P., "A Formal Approach to Learning from Examples," Proceedings of the 10th International Joint Conference on Artificial Intelligence, Milan, Italy, August 23–28, 1987.

Denning, Peter J., "The Science of Computing," American Scientist, Volume 74, January–February 1986.

DeWalt, Michael, Avionics Training Seminar, Kansas City, MO, December 8–9, 1992.

DOD-STD-2167A, "Defense System Software Development," February 29, 1988.

Donker, J. C., "Reasoning with Uncertain and Incomplete Information in Aerospace Applications," AGARD Conference Proceedings 499: Machine Intelligence for Aerospace Electronic Systems, Advisory Group for Aerospace Research and Development, Neuilly Sur Seine, France, May 1991.

Dreyfus, Hubert and Stuart Dreyfus, "Why Computers May Never Think Like People," Technology Review, January 1986.

Dupuy, Serge, "Contribution of Expert Systems to Avionics Advance Application on AS30 Laser Weapon System," Software Engineering and its Application to Avionics, AGARD-CP-439, Advisory Group for Aerospace Research and Development, Neuilly Sur Seine, France, November 1988.

Eimer, Erhard O., "Decision Aids: Disasters Waiting to Happen?" Proceedings of the IEEE 1987 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1987.

Elwell, D., et al., Avionic Data Bus Integration Technology, DOT/FAA/CT-91/19, U.S. Department of Transportation, Federal Aviation Administration, December 1991.

Elwell, D. and N. VanSuetendael, Software Quality Metrics, DOT/FAA/CT-91/1, U.S. Department of Transportation, Federal Aviation Administration, 1991.

Endres, Gunther, "Towards the 'Intelligent' Aircraft," Interavia Aerospace Review, Volume 46, April 1991.

Engelmore, Robert and Tony Morgan, Blackboard Systems, Addison-Wesley Publishing Company, New York, NY, 1988.

Farrell, Robert, "Intelligent Case Selection and Presentation," Proceedings of the 10th International Joint Conference on Artificial Intelligence, Milan, Italy, August 23–28, 1987.

Fey, J., et al., "Expert System for the Tornado Ground-Based Check-Out System," AGARD Conference Proceedings 499: Machine Intelligence for Aerospace Electronic Systems, Advisory Group for Aerospace Research and Development, Neuilly Sur Seine, France, May 1991.

Feyock, Stefan and Stamos Karamouzis, "Design of an Intelligent Information System for In-Flight Emergency Assistance," 1991 Goddard Conference on Space Applications of Artificial Intelligence, NASA Conference Proceeding CP-3110, May 1991.

Feyock, Stefan and Dalu Li, "Simulation-Based Reason about the Physical Propagation of Fault Effects," 1990 Goddard Conference on Space Applications of Artificial Intelligence, NASA Conference Proceeding CP-3068, May 1990.

Fikes, Richard and Tom Kehler, "The Role of Frame-Based Representation in Reasoning," Communications of the ACM, Volume 28, No. 9, September 1985.

A Framework for Embedded Avionics Expert Systems, Boeing Advanced Systems Public Relations Sales Brochure, Seattle, WA.

Frankovich, Ken, Ken Pedersen, and Stanley Bernsteen, "Expert System Applications to the Cockpit of the '90s," Proceedings of the IEEE 1985 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1985.

French, Scott W. and David O. Hamilton, Expert Systems V&V Guidelines Workshop, February 1992.

Friedland, Peter, "Special Section on Architectures for Knowledge-Based Systems," Communications of the ACM, Volume 28, No. 9, September 1985.

Frisch, Alan M., "Inference Without Chaining," Proceedings of the 10th International Joint Conference on Artificial Intelligence, Milan, Italy, August 23–28, 1987.

Frueh, George T., "Semiconductor Highlight," Electronic Component News, Volume 37, No. 7, July 1993.

The Fuzzy Source, Volume 2, Issue 1, Togai Infralogic, Inc., Irvine, CA, 1992.

Gabriel, Richard F., "Cockpit Automation," Kim M. Cardosi and M. Stephen Huntley eds., Human Factors for Flight Deck Certification Personnel, Draft, DOT-VNTSC-FAA-93-4, U.S. Department of Transportation, Research and Special Programs Administration, May 1993.

Galdes, Deborah K. and Philip J. Smith, "Building an Intelligent Tutoring System: Some Guidelines from a Study of Human Tutors," Proceedings of the Human Factors Society 34th Annual Meeting, Santa Monica, CA, 1990.

Geddes, Norman D., "Verification and Validation Testing of the Pilot's Associate," IEEE/AIAA 10th Digital Avionics Systems Conference Proceedings, IEEE Service Center, Piscataway, NJ, 1991.

Genesereth, Michael R. and Matthew L. Ginsberg, "Logic Programming," Communications of the ACM, Volume 28, No. 9, September 1985.

Gibbons, Greg D., et al., "Expert Systems: They Ain't What They Used to Be!" Proceedings of the IEEE 1988 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1988.

Gibbons, Greg D., Jonathan S. Abel, and Jill V. Josselyn, "ASAP: AI-Based Situation Assessment and Planning," Proceedings of the IEEE 1988 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1988.

Glickstein, Ira, Steve Ruberg, and Lt. John Marsh, "Database Management for Integrated Avionics System," Proceedings of the National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1992.

Glover, Richard D., "Application Experience with the NASA Aircraft Interrogation and Display System: A Ground-Support Equipment for Digital Flight Systems," IEEE/AIAA 5th Digital Avionics Systems Conference, Seattle, WA, October 31–November 3, 1983.

Glover, Richard D. and Richard R. Larson, "A Knowledge Based Application of the Extended Aircraft Interrogation and Display System," NASA Technical Memorandum 4327, Dryden Flight Research Facility, Edwards, CA, October 1991.

Gordon, Sallie E., Rhonda A. Kinghorn, and Kim A. Schmierer, "Representing Expert Knowledge for Instructional System Design: A Case Study," Proceedings of the Human Factors Society 35th Annual Meeting, Santa Monica, CA, 1991.

Gordon, Sallie E. and Vickie Lewis, "Knowledge Engineering for Hypertext Instructional Systems," Proceedings of the Human Factors Society 34th Annual Meeting, Santa Monica, CA, 1990.

Graham, Joyce M., "An Intelligent Spatial Database System for Interaction with a Real-Time Piloting Expert System," Proceedings of the IEEE 1987 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1987.

Green, Christopher J. R., "On the Use of Requirements in the Development of Knowledge-Based Systems," Uma G. Gupta ed., Validating and Verifying Knowledge-Based Systems, IEEE Computer Society Press, Los Alamitos, CA, 1991.

Green, Christopher J. R. and Marlene M. Keyes, "Verification and Validation of Expert Systems," Uma G. Gupta ed., Validating and Verifying Knowledge-Based Systems, IEEE Computer Society Press, Los Alamitos, CA, 1991.

Grimshaw, Capt. Jeffrey D. and Craig S. Anken, "A Distributed Environment for Testing Cooperating Expert Systems," AGARD Conference Proceedings 499: Machine Intelligence for Aerospace Electronic Systems, Neuilly Sur Seine, France, September 1991.

Hall, Lawrence O. and Abraham Kandel, "The Evolution from Expert Systems to Fuzzy Expert Systems," Fuzzy Expert Systems, Abraham Kandel ed., CRC Press, Boca Raton, FL, 1992.

Hammer, John, M., "Verification and Validation of Knowledge Bases in Associate Systems," DARPA Symposium on Associate Technology, Fairfax, VA, June 6–7, 1991.

Hammerstrom, Dan, "Working with Neural Networks," IEEE Spectrum, Volume 30, No. 7, July 1993.

Harvey, David S., "Artificial Intelligence in Combat Avionics," Avionics, Volume 17, No. 1, January 1993.

Hayes-Roth, Frederick, "Rule-Based Systems," Communications of the ACM, Volume 28, No. 9, September 1985.

Hertz, John, Anders Krogh, and Richard G. Palmer, Introduction to the Theory of Neural Computation, Addison Wesley Publishing Company, Redwood City, CA, 1991.

Hillman, David V., "Integrating Neural Nets and Expert Systems," AI Expert, June 1990.

Hillman, Donald J., "Artificial Intelligence," Human Factors, Volume 27, No. 1, 1985.

Holla, K. and B. Benninghofen, "A Threat Management System," AGARD Conference Proceedings 499: Machine Intelligence for Aerospace Electronic Systems, Advisory Group for Aerospace Research and Development, Neuilly Sur Seine, France, September 1991.

Hui, Patrick J. and Amiya R. Nayak, "EXNAV: An Intelligent Sensor Integrator," Proceedings of the IEEE 1988 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1988.

Humphrey, Timothy L., "Using PROLOG in Natural Language Systems," Proceedings of the IEEE 1985 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1985.

Hunter, Lawrence and David J. States, "Bayesian Classification of Protein Structure," IEEE Expert, Volume 7, No. 4, August 1992.

The Huntington Group, Huntington Technical Brief, No. 34, Menlo Park, CA, January 1993.

Iline, Herman and Henry Kanoui, "Extending Logic Programming to Object Programming: The System LAP," Proceedings of the 10th International Joint Conference on Artificial Intelligence, Milan, Italy, August 23–28, 1987.

Improving Aircraft Safety, National Academy of Sciences, Washington, DC, 1980.

Intel[1] Corporation, 80170NW - Electrically Trainable Analog Neural Network, Santa Clara, CA, May 1990.

Intel[2] Corporation, "New Wave" Computing? – An Introduction to Neural Networks, November 1990.

Iorgulescu, Daniela T., David F. Giere, and Carol S. Giffen, "Artificial Expertise in Systems Engineering," IEEE/AIAA 10th Digital Avionics Systems Conference Proceedings, IEEE Service Center, Piscataway, NJ, 1991.

Irwin, J. David, Basic Engineering Circuit Analysis, 2nd ed., Macmillan Publishing Company, New York, NY, 1987.

Jafar, Musa and A. Terry Bahill, "Interactive Verification of Knowledge-Based Systems," IEEE Expert, Volume 8, No. 1, 1993.

Janowitz, Joan, Handbook–Volume III Digital Systems Validation Book Plan, DOT/FAA/CT-93/16, U.S. Department of Transportation, Federal Aviation Administration, July 1993.

Johnson, Sally C. and Ricky W. Butler, "Design for Validation," IEEE/AIAA 10th Digital Avionics Systems Conference Proceedings, IEEE Service Center, Piscataway, NJ, 1991.

Johnson, William B., "Advanced Technology for Aviation Maintenance Training: An Industry Status Report and Development Plan," Proceedings of the Human Factors Society 34th Annual Meeting, Santa Monica, CA, 1990.

Johnson, William B. and Jeffrey E. Norton, "Integrated Systems for Training, Aiding, and Information Retrieval," Proceedings for the East–West Conference in Emerging Computer Technology in Education, Moscow, Russia, 1992.

Kang, Yue and A. Terry Bahill, "A Tool for Detecting Expert System Errors," Uma G. Gupta ed., Validating and Verifying Knowledge-Based Systems, IEEE Computer Society Press, Los Alamitos, CA, 1991.

Kartalopoulos, Stamatios V., "A Plateau of Performance?," IEEE Communications, Volume 30, No. 9, September 1992.

Kearsley, G. P., ed., Artificial Intelligence and Instruction, Addison-Wesley Publishing Company, Reading, MA, 1987.

Keller, Forrest L., "Basic Electronic Systems Certification," IEEE/AIAA 11th Digital Avionics Systems Conference Proceedings, IEEE Service Center, Piscataway, NJ, 1992.

Khaksari, Gholam H., "Expert Diagnostic System," The First International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems, June 1-3, 1988.

Klein, Gary and Roberta Calderwood, "Human Factors Considerations for Expert Systems," Proceedings of the IEEE 1986 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1986.

Klimasauskas, Casimir C., "Neural Networks: An Engineering Perspective," IEEE Communications, Volume 30, No. 9, September 1992.

Kosko, Bart, Neural Networks and Fuzzy Systems, Prentice Hall Publishing Company, Englewood Cliffs, NJ, 1992.

Krogmann, Uwe K., "Introduction to Neural Computing and Categories of Neural Network Applications to Guidance, Navigation, and Control," AGARD Lecture Series 179: Artificial Neural Network Approaches in Guidance and Control, Advisory Group for Aerospace Research and Development, Neuilly Sur Seine, France, September 1991.

Lawler, R. W. and M. Yazdani, eds., Artificial Intelligence and Education Volume 1: Learning Environments and Tutoring Systems, Ablex, Norwood, NJ, 1987.

Lawrence, Jeannette, Introduction to Neural Networks, Sylvia Ludeking ed., California Scientific Software, 1991.

Layer, Kevin D. and Chris Richardson, "LISP," Communications of the ACM, Volume 34, No. 9, September 1991.

Lea, Robert N., "Application of Fuzzy Sets to Rule-Based Expert System Development," Telematics and Informatics, Volume 6, Nos. 3 and 4, 1989.

Leavitt, C. A. and D. M. Smith, "Integrated Dynamic Planning in the Pilot's Associate," AIAA Guidance, Navigation, and Control Conference, Part 1, Boston, MA, August 14–16, 1989.

Lee, Chuen-Chien, "A Self-Learning Rule-Based Controller Employing Approximate Reasoning and Neural Net Concepts," International Journal of Intelligent Systems, Volume 6, 1991.

Leeper, Kenneth R., "Artificial Intelligence Programming in Ada," IEEE/AIAA 9th Digital Avionics Systems Conference Proceedings, IEEE Service Center, Piscataway, NJ, 1990.

Legg, Gary, "Special Tools and Chips Make Fuzzy Logic Simple," EDN, Volume 37, No. 14, July 6, 1992.

Legg, Gary, "Microcontrollers Embrace Fuzzy Logic," EDN, Volume 38, No. 19, September 1993.

Leinweber, David and David Carleton, "The PICON Real-Time Expert System Tool," Proceedings of the IEEE 1986 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1986.

Lenorovitz, David R. and Ray A. Reaux, "Integrating Human Factors Guidance Information within the USI Design/Rapid Prototyping Process," Proceedings of the IEEE 1986 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1986.

Level5 Object User's Guide, Information Builders, Inc., New York, NY, 1992.

Levi, Keith R., et al., "An Explanation-Based-Learning Approach to Knowledge Compilation," IEEE Expert, Volume 7, No. 3, June 1992.

Levine, Robert I., Diane E. Drang, and Barry Edelson, A Comprehensive Guide to AI and Expert Systems, McGraw-Hill Publishing Company, New York, NY, 1986.

Lim, Ee-Peng and Vladimir Cherkassky, "Semantic Networks and Associative Databases," IEEE Expert, Volume 7, No. 4, August 1992.

Lischke, Michael P. and Kenneth L. Meyer, "TEAMS: Technical Aircraft Expert Maintenance System," Proceedings of the IEEE 1992 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1992.

Lizza, Carl, Sheila Banks, and Michael A. Whelan, "Pilot's Associate: Evolution of a Functional Prototype," AGARD Conference Proceedings 499: Machine Intelligence for Aerospace Electronic Systems, Advisory Group for Aerospace Research and Development, Neuilly Sur Seine, France, September 1991.

Lizza, Carl and Carl Friedlander, "The Pilot's Associate: A Forum for the Integration of Knowledge Based Systems and Avionics," Proceedings of the IEEE 1988 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1988.

Lovesey, E. J. and R. I. Davis, "Integrating Machine Intelligence into the Cockpit to Aid the Pilot," AGARD Conference Proceedings 499: Machine Intelligence for Aerospace Electronic Systems, Advisory Group for Aerospace Research and Development, Neuilly Sur Seine, France, September 1991.

MacDonald, Jim and Klaus K. Obermeier, "Towards a Taxonomy for Knowledge Representation Schemas," Proceedings of the IEEE 1986 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1986.

Madni, Azad M., "HUMANE: A Knowledge-Based Simulation Environment for Human-Machine Function Allocation," Proceedings of the IEEE 1988 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1988.

Manheimer, Jerry M. and Thomas A. Kanarski, "The Application of Psychological Scaling Techniques in Modeling Expert Knowledge," Proceedings of the IEEE 1986 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1986.

Marcot, Bruce, "Testing Your Knowledge Base," Uma G. Gupta ed., Validating and Verifying Knowledge-Based Systems, IEEE Computer Society Press, Los Alamitos, CA, 1991.

Marcus, Sandra, "Salt: A Knowledge Acquisition Tool that Checks and Helps Test a Knowledge Base," Uma G. Gupta ed., Validating and Verifying Knowledge-Based Systems, IEEE Computer Society Press, Los Alamitos, CA, 1991.

Masotto, Tom, Carol Babikyan, and Richard Harper, "Knowledge Representation into Ada Parallel Processing," NASA Contractor Report 187451, NASA Langley Research Center, Hampton, VA, July 1990.

McCay, Scott, "CLIM: The Common Lisp Interface Manager," Communications of the ACM, Volume 34, No. 9, September 1991.

McCoy, Michael S. and Randy M. Boys, "Human Performance Models Applied to Intelligent Decision Support Systems," Proceedings of the IEEE 1987 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1987.

McManus, John W., "Design and Analysis Tools for Concurrent Blackboard Systems," IEEE/AIAA 10th Digital Avionics Systems Conference Proceedings, IEEE Service Center, Piscataway, NJ, 1991.

McNeese, Michael D., "Humane Intelligence: A Human Factors Perspective for Developing Intelligent Cockpits," Proceedings of the IEEE 1986 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1986.

McNeese, Michael D. and Brian S. Zaff, "Knowledge as Design: A Methodology for Overcoming Knowledge Acquisition Bottlenecks in Intelligent Interface Design," Proceedings of the Human Factors Society 35th Annual Meeting, Santa Monica, CA, 1991.

McNulty, Christa, "Knowledge Engineering for a Piloting Expert System," Proceedings of the IEEE 1987 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1987.

Mehrotra, Mala, "Rule Groupings: A Software Engineering Approach Towards Verification of Expert Systems," NASA Contractor Report 4372, NASA Langley Research Center, Hampton, VA, May 1991.

Mehrotra, Mala and Chris Wild, "Multi-Viewpoint Clustering Analysis," Workshop on Verification, Validation, and Testing of Intelligent Systems, 9th IEEE Conference on Artificial Intelligence for Applications, Orlando, FL, March 2, 1993.

Minges, Mark E., "Integrated Communications, Navigation, Identification, Avionics (ICNIA) Expert Systems for Fault Tolerant Avionics," AGARD Conference Proceedings 499: Machine Intelligence for Aerospace Electronic Systems, Advisory Group for Aerospace Research and Development, Neuilly Sur Seine, France, September 1991.

Mitsubishi Electric, Product Literature, Japan, 1992.

Mitta, Deborah, Newton C. Ellis, and Dick B. Simmons, "Human Factors Data: Knowledge Sources for Intelligent Design Associates," Proceedings of the Human Factors Society 34th Annual Meeting, Santa Monica, CA, 1990.

Miyamoto, Sadaaki, Fuzzy Sets in Information Retrieval and Cluster Analysis, Kluwer Academic Publishers, Boston, MA, 1990.

Morley, Richard E. and William A. Taylor, "What is Artificial Intelligence?," Digital Design, April 1986.

Morley, Richard E. and William A. Taylor, "Artificial Intelligence Basics: Hardware Follows Software," Digital Design, May 1986.

Morley, Richard E. and William A. Taylor, "Why Bother with Expert Systems?," Digital Design, July 1986.

Muller, Hans, "LispView: Leverage Through Integration," Communications of the ACM, Volume 34, No. 9, September 1991.

Murphy, Thomas, "AI Apprentice," AI Expert, Volume 8, No. 4, April 1993.

Naser, Joseph A., "Nuclear Power Plant Expert System Verification and Validation," Uma G. Gupta ed., Validating and Verifying Knowledge-Based Systems, IEEE Computer Society Press, Los Alamitos, CA, 1991.

The National Plan for Aviation Human Factors, Volume 1, Draft, U.S. Department of Transportation, Federal Aviation Administration, April 1991.

The National Plan for Aviation Human Factors, Volume 2, Draft, U.S. Department of Transportation, Federal Aviation Administration, November 1990.

Nelson, Dale E. and Steven K. Rogers, "A Taxonomy of Neural Network Optimality," Proceedings of the IEEE 1992 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1992.

NeuroDynamx Literature, NDX Neural Accelerators, NeuroDynamx, Inc., Boulder, CO, 1994.

Ng, Andrew, "A Cooperative Expert System Architecture for Embedded Avionics," Proceedings of the IEEE 1988 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1988.

Nilsson, Nils J., Principles of Artificial Intelligence, Tioga Publishing Company, Palo Alto, CA, 1980.

Nordwall, Bruce D., "Cockpit Safety Researchers Eye Verbal Advice System," Aviation Week & Space Technology, Volume 137, No. 14, October 5, 1992.

Normile, Dennis, "Mimicking the Brain," Popular Science, November 1992.

Norton, J. E. and W. B. Johnson, "Microcomputer Intelligence for Technical Training (MITT): The Evolution of an Intelligent Tutoring System," Proceedings, NASA 1991 Conference on Intelligent Computer-Aided Training, Houston, TX, 1991.

Obermeier, Klaus K. and Janet J. Barron, "Time to Get Fired Up," Byte, August 1989.

O'Keefe, Robert M., Osman Balci, and Eric P. Smith, "Validating Expert System Performance," Uma G. Gupta ed., Validating and Verifying Knowledge-Based Systems, IEEE Computer Society Press, Los Alamitos, CA, 1991.

O'Keefe, Robert M. and Daniel O'Leary, "Expert System Verification and Validation: A Survey and Tutorial," Workshop on Verification, Validation, and Testing of Intelligent Systems, 9th IEEE Conference on Artificial Intelligence for Applications, Orlando, FL, March 2, 1993.

O'Leary, Daniel E., "Measuring the Quality of Computer Model Performance," Workshop on Verification, Validation, and Testing of Intelligent Systems, 9th IEEE Conference on Artificial Intelligence for Applications, Orlando, FL, March 2, 1993.

Omron Electronics, Inc., "An Introduction to Fuzzy Logic and its Application in Control Systems," Fuzzy Logic, a 21st Century Technology, 1991.

Ostgaard, John C. and D. Reed Morgan, "PAVE PILLAR and PAVE PACE: Avionics System Architecture for the 21st Century," ERA Seminar Proceedings: Military Avionics Architectures for Today and Tomorrow, ERA Report No. 88-0437, ERA Technology Ltd., Surrey, England, 1989.

Ovenden, C. R., "AI Applications to Tactical Decision Aids," IEEE/AIAA 8th Digital Avionics Systems Conference Proceedings, IEEE Service Center, Piscataway, NJ, 1988.

Palko, Katherine and Randy Boys, "Augmenting the Pilot/Vehicle Interface Through Capability Assessment," Proceedings of the IEEE 1987 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1987.

Papp, M. L., W. R. Braisted, and R. F. Taylor, "A Prototype Expert System for Analysis of Turbine Engine Components," Proceedings of the IEEE 1992 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1992.

Pearce, Michael, et al., "Case-Based Design Support," IEEE Expert, 1992.

Pederson, Ken, "Part I: The Well-Structured Rule, the Basic Building Block for Knowledge Bases," Uma Gupta ed., Validating and Verifying Knowledge-Based Systems, IEEE Computer Society Press, Los Alamitos, CA, 1989.

Pederson, Ken, "Well-Structured Knowledge Bases: Part II," Uma Gupta ed., Validating and Verifying Knowledge-Based Systems, IEEE Computer Society Press, Los Alamitos, CA, 1989.

Pederson, Ken, "Well-Structured Knowledge Bases: Part III," Uma Gupta ed., Validating and Verifying Knowledge-Based Systems, IEEE Computer Society Press, Los Alamitos, CA, 1989.

Penn, Brian S., "Design Considerations in the Development of the Meta-Expert System," Proceedings of the IEEE 1986 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1986.

Perlin, Mark, Revision of "An OPS5 Primer: Introduction to Rule-Based Expert Systems," by Porter D. Sherman and John C. Martin, IEEE Communications, Volume 31, No. 1, January 1993.

Piers, M. A. and J. C. Donker, "A Knowledge Based Assistant for Diagnosis in Aircraft Maintenance," AGARD Conference Proceedings 449: Machine Intelligence for Aerospace Electronic Systems, Advisory Group for Aerospace Research and Development, Neuilly Sur Seine, France, September 1991.

Pilet, S. C. and R. O. Stenerson, "Avionics Expert Systems: The Transition to Embedded Systems," Proceedings of the IEEE 1987 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1987.

Plant, Robert T., "Pattern Directed Inference Systems: A Development Methodology," Workshop on Verification, Validation, and Testing of Intelligent Systems, 9th IEEE Conference on Artificial Intelligence for Applications, Orlando, FL, March 2, 1993.

Plant, Robert T. and Panagiotis Tsoumpas, "An Integrated Methodology for Knowledge-Based System Development," Workshop on Verification, Validation, and Testing of Intelligent Systems, 9th IEEE Conference on Artificial Intelligence for Applications, Orlando, FL, March 2, 1993.

Plutowski, Mark, "VOSS-VHSIC Hosted Expert Systems," Proceedings of the IEEE 1986 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1986.

Pohlmann, Lawrence D. and J. Roland Payne, "Pilot's Associate Demonstration One: A Look Back and Ahead," Proceedings of the IEEE 1986 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1986.

Polson, Martha Campbell, "Status and Future Directions of Intelligent Tutoring Systems." Proceedings of the Human Factors Society 33rd Annual Meeting, Santa Monica, CA, 1989.

Prevot, T., R. Onken, and H. L. Dudek, "Knowledge-Based Planning for Controlled Airspace Flight Operation as Part of a Cockpit Assistant," AGARD Conference Proceeding 504: Air Vehicle Mission Control and Management, Advisory Group for Aerospace Research and Development, Neuilly Sur Seine, France, October 1991.

Psotka, Joseph, "Advancing the Mind/Machine Interface: Qualitative Simulations, Hypertext, and Natural Language Processing," Proceedings of the Human Factors Society 33rd Annual Meeting–1989, Santa Monica, CA, 1989.

Pukite, P. R., J. Pukite, and D. S. Barnhart, "Expert System for Redundancy and Reconfiguration Management," Proceedings of the IEEE 1992 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1992.

Rao, Ming and Tsung-Shann Jiang, "A New Method to Design Intelligent Control Systems," Proceedings of the IEEE 1988 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1988.

Rich, Elaine and Kevin Knight, Artificial Intelligence, 2nd ed., McGraw-Hill Publishing Company, New York, NY, 1991.

Roberts, K., "TACAID – A Knowledge Based System for Tactical Decision Making," AGARD Conference Proceedings 499: Machine Intelligence for Aerospace Electronic Systems, Advisory Group for Aerospace Research and Development, Neuilly Sur Seine, France, September 1991.

Rocha, A. F., et al., "The Physiology of the Expert System," Fuzzy Expert Systems, Abraham Kandel ed., CRC Press, Boca Raton, FL, 1992.

Rock, Denny, Don Malkoff, and Ron Stewart, "AI and Aircraft Health Monitoring," AI Expert, February 1993.

Rolston, David W., Principles of Artificial Intelligence and Expert Systems Development, McGraw-Hill Publishing Company, New York, NY, 1988.

Roth, Al, "The Practical Application of PROLOG," AI Expert, Volume 8, No. 4, April 1993.

Rothman, Michael J., "IJCNN '92," IEEE Expert, Volume 8, No. 1, February 1993.

Rouse, William B., Norman D. Geddes, and Renwick E. Curry, "An Architecture for Intelligent Interfaces: Outline of an Approach to Supporting Operators of Complex Systems," Proceedings of the IEEE 1986 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1986.

Rouse, William B., Norman D. Geddes, and John M. Hammer, "Computer-Aided Fighter Pilots," IEEE Spectrum, Volume 27, No. 3, March 1990.

RTCA/DO-160C, "Environmental Conditions and Test Procedures for Airborne Equipment," Radio Technical Commission for Aeronautics, Washington, DC, December 1989.

RTCA/DO-178B, "Software Considerations in Airborne Systems and Equipment Certification," Radio Technical Commission for Aeronautics, Washington, DC, December 1992.

RTCA Paper No. 548-93/SC180-18, Radio Technical Commission for Aeronautics, Washington, DC, February 2, 1994.

Rudolph, F. M., D. A. Homoki, and G. A. Sexton, "Diverter' Decision Aiding for In-Flight Diversions," NASA Contractor Report 182070, NASA Langley Research Center, Hampton, VA, August 1990.

Rushby, John, "Quality Measures and Assurance for AI Software," NASA Contractor Report 4187, NASA Langley Research Center, Hampton, VA, October 1988.

Rushby, John and Judith Crow, "Evaluation of an Expert System for Fault Detection, Isolation, and Recovery in the Manned Maneuvering Unit," NASA Contractor Report 187466, NASA Langley Research Center, Hampton, VA, December 1990.

Rushby, John and R. Alan Whitehurst, "Formal Verification of AI Software," NASA Contractor Report 181827, NASA Langley Research Center, Hampton, VA, February 1989.

Ryder, Joan M., et al., "An Integrated Embedded Training and Decision Aiding Design Methodology," Proceedings of the Human Factors Society 34th Annual Meeting, Santa Monica, CA, 1990.

Sadeghi, Tom and Gerry Mayville, "Fault-Tolerant, Flight-Critical Control Systems," AGARD Conference Proceedings 456: Fault Tolerant Design Concepts for Highly Integrated Flight Critical Guidance and Control Systems, Advisory Group for Aerospace Research and Development, Neuilly Sur Seine, France, October 13, 1989.

Savant, Jr., C. J., Martin S. Roden, and Gordon L. Carpenter, Electronic Circuit Design, An Engineering Approach, The Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA, 1987

Schaefer, B. A., et al., "A Knowledge-Based Intelligent Tutoring System: ACQUIRE^{TM}-ITS," AGARD Conference Proceedings 499: Machine Intelligence for Aerospace Electronic Systems, Advisory Group for Aerospace Research and Development, Neuilly Sur Seine, France, September 1991.

Schultz, Roger D. and James R. Geissman[1], "Bridging the Gap Between Static and Dynamic Verification," Uma G. Gupta ed., Validating and Verifying Knowledge-Based Systems, IEEE Computer Society Press, Los Alamitos, CA, 1991.

Schultz, Roger D. and James R. Geissman[2], "Verification and Validation of Expert Systems," Uma G. Gupta ed., Validating and Verifying Knowledge-Based Systems, IEEE Computer Society Press, Los Alamitos, CA, 1991.

Schutte, Paul C. and Kathy H. Abbott, "An Artificial Intelligence Approach to Onboard Fault Monitoring and Diagnosis for Aircraft Applications," AIAA Guidance and Control Conference, Williamsburg, VA, 1986.

Schwartz, Daniel G. and George J. Klir, "Fuzzy Logic Flowers in Japan," IEEE Spectrum, Volume 29, No. 7, July 1992.

Schwartz, Tom J., "IEEE Gets Fuzzy," IEEE Expert, Volume 7, No. 3, June 1992.

Segre, Alberto Maria, "Applications of Machine Learning," IEEE Expert, Volume 7, No. 3, June 1992.

Seidman, Abraham N., "Neural Networks and Digital Avionics," IEEE/AIAA/NASA Digital Avionics Systems Conference Proceedings, IEEE Service Center, Piscataway, NJ, 1990.

Selcon, Stephen J., "Decision Support in the Cockpit: Probably a Good Thing?" Proceedings of the Human Factors Society 34th Annual Meeting, Santa Monica, CA, 1990.

Semple, W. G., "Development of Tactical Decision Aids," AGARD Conference Proceedings 499: Machine Intelligence for Aerospace Electronic Systems, Advisory Group for Aerospace Research and Development, Neuilly Sur Seine, France, September 1991.

Sewell, Daniel R. and William B. Johnson, "The Effects of Rapid Prototyping on User Behavior on System Design," Journal of the Washington Academy of Sciences, Volume 80, No. 2, June 1991.

Shaw^1, Julie, "Neural Network Resource Guide," AI Expert, Volume 8, No. 2, February 1993.

Shaw^2, Julie, "AI Language Resource Guide," AI Expert, Volume 8, No. 4, April 1993.

Shear, David, "Neural Network and Fuzzy Logic Combine to Create COP8 Code," EDN, Volume 38, No. 11, May 27, 1993.

Shelnutt, Jack B., et al., "Pilot's Associate Demonstration One: A Look Inside," Proceedings of the IEEE 1986 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1986.

Sheppard, John W. and William R. Simpson, "Expert Systems for Diagnostic Testing," Avionics, October 1992.

Shepherd, William T. and William B. Johnson, "Aircraft Maintenance Challenges and Human Factors Solutions," Seminar on Flight Safety and Human Factors, International Civil Aviation Organization, November 1991.

Shewhart, Mark, "Interpreting Statistical Process Control (SPC) Charts Using Machine Learning and Expert System Techniques," Proceedings of the IEEE 1992 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1992.

Silbert, Mark, et al., "A Tool for Development of AI Hybrid Systems," IEEE/AIAA Digital Avionics Systems Conference Proceedings, IEEE Service Center, Piscataway, NJ, 1986.

Simpson, Patrick K., "Neural Network Paradigms," AGARD Lecture Series 179: Artificial Neural Network Approaches in Guidance and Control, Advisory Group for Aerospace Research and Development, Neuilly Sur Seine, France, September 1991.

Sitz, Joel R. and Todd H. Vernon, "Flight Control System Design Factors for Applying Automated Testing Techniques," NASA Technical Memorandum 4242, October 1990.

Small, Ronald A. and Charles W. Howard, "A Real-Time Approach to Information Management in a Pilot's Associate," IEEE/AIAA 10th Digital Avionics Systems Conference Proceedings, IEEE Service Center, Piscataway, NJ, 1991.

Smith, Carolyn and Horace Sklar, "Embedded Expert Systems for Fault Detection and Isolation," IEEE/AIAA Digital Avionics Systems Conference Proceedings, IEEE Service Center, Piscataway, NJ, 1986.

Sorrells, Mark E., "A Time-Constrained Inference Strategy for Real Time Expert Systems," Proceedings of the IEEE 1985 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1985.

Steele, Jr., Guy L., Common LISP: The Language, Digital Press, 1984.

Tazelaar, Jane Morrill, "Neural Networks," Byte, August 1989.

Texas Instruments Literature, "Fuzzy Logic Moves into DSP Design Arena," Integration, North American Edition, Volume 11, No. 1, Texas Instruments, 1994.

Togai, Masaki and Horoyuki Watanabe, "Expert System on a Chip: An Engine for Approximate Reasoning," Fuzzy Expert Systems, Abraham Kandel ed., CRC Press, Boca Raton, FL, 1992.

Touretzky, David S. and Dean A. Pomerleau, "What's Hidden in the Hidden Layers?," Byte, August 1989.

Valstar, Jacob E., "Expert System Development on a Spreadsheet," Proceedings of the IEEE 1986 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1986.

Waibel, Alex and John Hampshire, "Building Blocks for Speech," Byte, August 1989.

Waldron, Vince, Harold W. Sharp, and Lt. Scott A. Stefanov, "Distributed Expert Management System," Proceedings of the IEEE 1987 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1987.

Wagner, Elaine A., "Onboard Automatic Aid and Advisory for Pilots of Control-Impaired Aircraft," Journal of Guidance, Control, and Dynamics, Volume 14, No. 4, July–August 1991.

Wang, David C. and James Thompson, "An Adaptive Data Sorter Based on Probabilistic Neural Networks," Topics in Engineering, Volume 3, AIL Systems, Inc., Deer Park, NY, 1992.

Warn, Keith, "Expert System Shell Standardization – The Controversy," Proceedings of the IEEE 1987 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1987.

Waterman, Donald A., A Guide to Expert Systems, Addison-Wesley Publishing Company, Reading, MA, 1986.

Weiss, Ray, "Fuzzy Coprocessor Performs up to 870,000 Evaluations per Second," EDN, Volume 38, No. 22, October 28, 1993.

Weiss, Sholom M. and Casimir A. Kulikowski, A Practical Guide to Expert Systems, Rowman and Allanheld Publishers, Totowa, NJ, 1984.

Wellens, Rodney A. and Michael D. McNeese, "A Research Agenda for the Social Psychology of Intelligent Machines," Proceedings of the IEEE 1987 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1987.

Wenger, E., Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge, Morgan Kauffmann, Los Altos, CA, 1987.

Whitaker, Leslie A., Richard H. Stottler, and James A. King, "Case-Based Reasoning: Taming the Similarity Heuristic," Proceedings of the Human Factors Society 34th Annual Meeting, Santa Monica, CA, 1990.

White, David J. and Elizabeth A. Bart, "An Expert System for VLSI Datapath Synthesis," Proceedings of the IEEE 1988 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1988.

Wickens, Christopher D., "Decision Making," Kim M. Cardosi and M. Stephen Huntley eds., Human Factors for Flight Deck Certification Personnel, Draft, DOT-VNTSC-FAA-93-4, U.S. Department of Transportation, Research and Special Programs Administration, May 1993.

Wickens, Christopher D., Engineering Psychology and Human Performance, 2nd ed., Harper Collins Publishing Company, 1992.

Widman, Lawrence E., Kenneth A. Loparo, and Norman R. Nielsen, Artificial Intelligence, Simulation, and Modeling, John Wiley and Sons Publishing Company, New York, NY, 1989.

Wiederholt, Bradley J., "MITT Writer: An Authoring System for Developing Intelligent Tutors for Complex Technical Domains," Galaxy Scientific Corporation, 1991.

Wiener, Earl L. and David C. Nigel, Human Factors in Aviation, Academic Press, Inc., San Diego, CA, 1988.

Wilber, George F. and E. Jean Dryer, "Strategic Real-Time Airborne Electronic Warfare Using Knowledge Base Techniques," Proceedings of the IEEE 1988 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1988.

Williams, Dr. R. M. and J. J. Davidson, "AI for RPVs, Sensor Driven Airborne Replanner, for a Robotic Aircraft Sensor Platform," AGARD Conference Proceedings 499: Machine Intelligence for Aerospace Electronic Systems, Advisory Group for Aerospace Research and Development, Neuilly Sur Seine, France, September 1991.

Williams, Tom, "Fuzzy Logic is Anything but Fuzzy," Computer Design, April 1992.

Winston, Patrick Henry, Artificial Intelligence 2nd Edition, Addison-Wesley Publishing Company, Reading, MA, 1984.

Wood, James, et al., "A Rule-Based Logic Design Assistant," Proceedings of the IEEE 1986 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1986.

Woolf, Beverly and Tom Murray, "A Framework for Representing Tutorial Discourse," Proceedings of the 10th International Joint Conference on Artificial Intelligence, Milan, Italy, August 23–28, 1987.

"Workshop on Verification, Validation, and Testing of Intelligent Systems," 9th IEEE Conference on Artificial Intelligence for Applications, Orlando, FL, March 2, 1993.

Xiaofeng, Li, "What's So Bad About Rule-Based Programming?," IEEE Software, September 1991.

Yen, Mike, "Lisp-to-Ada Reengineering Issues and Support Environments for Fielding Real-Time Systems," IEEE/AIAA 11th Digital Avionics Systems Conference Proceedings, IEEE Service Center, Piscataway, NJ, 1992.

Yops, Jr., Thomas E. and John R. Moore, "An Integrated Vehicle Management System Concept for Military Transports," Proceedings of the IEEE 1986 National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1986.

Young, Christina M., "Fuzzy Controller for a Pitot-Static Test Set," National Aerospace and Electronics Conference, IEEE Service Center, Piscataway, NJ, 1993.

Ziemacki, Mike, "Fuzzy Logic Microcontroller," ECN, Volume 37, No. 3, March 1993.

GLOSSARY

A PRIORI. Pertaining to deductive reasoning from assumed axioms or supposedly self-evident principles, without reference to experience.

ADAPTIVE FUZZY SYSTEMS. Fuzzy controllers with the ability to learn and explain their reasoning.

ALGORITHM. A set of well-defined rules for the solution of a problem in a finite number of steps.

ALPHA-CUT. A family of crisp sets that is used to interpret the membership function by a fuzzy controller.

ANALOG. Pertaining to devices, data, circuits, or systems that operate with variables that are presented by continuously measured voltages or other quantities.

ARC. The part of a semantic network that connects one object to another.

ARTIFICIAL INTELLIGENCE. The subfield of computer science concerned with emulating human intelligence by use of software and hardware techniques.

ASSISTANT. An Expert System that performs a technically limited subset of an expert's task.

ASSOCIATIVE MEMORY NETWORK. A network where different input patterns, if sufficiently similar, become associated with one another (trigger the same response).

ASYNCHRONOUS. Operating at a speed determined by the circuit functions rather than by timing signals.

ATTRACTOR. The geometrical pattern toward which the trajectory of a dynamical system, presented by a curve in phase space, converges in the course of time.

AXON. The pathway that carries the output of a biological neuron.

BACK PROPAGATION. An algorithm used to adjust weights connecting neurons in successive layers of multi-layer networks. Back propagation learns by example and repetition.

BACKWARD CHAINING. A method of problem solving that works backward from a known conclusion in an attempt to find a path of reasoning in order to justify the conclusion.

BASIN OF ATTRACTION. The collection of all possible initial conditions of a dynamical system for which the trajectories representing that system in phase space will converge to a particular attractor.

BINARY. Composed of or characterized by two parts or elements.

BLACK BOX. A component having known input and output, that can be readily inserted into or removed from a specific place in a larger system without knowledge of the component's detailed internal structure.

BLACKBOARD. The controlling agent in a blackboard system that tracks activity between the knowledge sources.

BLACKBOARD SYSTEM. An Expert System that uses multiple knowledge sources that share data.

BOOLEAN LOGIC. The basic mathematics needed to study the logic design of digital systems. Boolean logic consists of two values and three basic operators.

BREADTH FIRST. The method of searching for solutions by considering all possible subgoals on one level before proceeding to the next level.

CERTAINTY FACTORS. Values that represent the level of belief associated with a fact or a rule.

CERTIFICATION. The process of obtaining FAA approval for the design, manufacture, and/or sale of aircraft and associated systems, subsystems, and parts.

CLOSED-LOOP. A family of automatic control units linked together with a process to form an endless chain; the effects of control action are constantly measured so that if the controlled quantity departs from the norm, the control units act to bring it back.

CLUSTERING. A technique to group similar objects together under the same label.

CODIFYING. A method of representing knowledge or data so they can be stored in a knowledge base or database for access by a computer.

COLLEAGUE. An Expert System that performs a significant subset of an expert's task.

CONNECTION WEIGHT. A method of representing the strength of the connection between nodes.

CONTROL SYSTEM. A system that regulates another system's behavior and adjusts it according to the desired behavior.

CONVENTIONAL PROGRAMMING. The use of standard programming languages, as opposed to application development languages, financial planning languages, query languages, and report programs.

CONVERGENCE. The point when the outputs of a Neural Network are no longer changing.

CRISP SET. The membership concept used in traditional system modelling, where objects are either in or out of the set with no intermediate step.

DATA DRIVEN. Describing the execution of a program in a data flow system, in which an instruction is carried out whenever all its input values are present.

DECLARATIVE. Generally, the knowledge found in the knowledge base; independent knowledge that does not contain procedures or methods of handling other knowledge.

DEDUCTIVE. Learning by logical inference or inferring from available knowledge.

DEFUZZIFICATION. The process of translating fuzzy, or non-crisp values into crisp, clearly defined values.

DEMON. A procedure activated by changing or accessing values in a database.

DENDRITE. The pathway that carries the input of a biological neuron.

DEPTH FIRST. A method of searching for a solution by pursuing each branch of the search tree to the end before considering another branch.

DOT PRODUCT. The inner product of vectors $(x_1, ..., x_n)$ and $(y_1, ..., y_n)$ from n-dimensional euclidean space is the sum of $s_i y_i$, as i ranges from 1 to n.

EMPIRICAL MODEL. The model of a system based on data and the principles of statistics. Empirical models produce relationships, not insight.

EVENT. A change to the blackboard of a Blackboard System.

EXPERT. 1. An Expert System that approaches an expert's level of performance within a given domain. 2. A person who has mastered solving specific types of problems.

EXPERT SYSTEM. A computer system designed to simulate the problem solving behavior of a human expert.

EXPERT SYSTEM SHELL. An Expert System development tool and run-time environment with a modifiable knowledge base.

FACETTE. The pieces of knowledge in a frame.

FEED FORWARD NEURAL NETWORK. A Neural Network where the only connections are from a neuron in one layer to a neuron in the next layer.

FILTER. A transmission network used in electrical systems for the selective enhancement of a given class of input signals.

FORWARD CHAINING. A method of solving problems by beginning with certain data and moving down the inference chain until a solution is reached.

FRAME. A way of representing Expert System knowledge that consists of a set of slots that contain data.

FRAME LABEL. A unique frame identification.

FUZZIFICATION. The process of converting crisp inputs into fuzzy values.

FUZZY CONTROL VARIABLE. A parameter whose value determines the action taken by fuzzy rules.

FUZZY EXPERT SYSTEM. An Expert System that uses fuzzy logic techniques to solve problems.

FUZZY LOGIC. The theory of representing vague or imprecise concepts to model inherent conditions.

FUZZY SET. An extension of the concept of a set, in which the characteristic function which determines membership of an object in the set can take on any value between 0 and 1.

FUZZY TERMS. Words used for descriptions that may not be precise in determining set memberships.

GRACEFUL DEGRADATION. A programming technique to prevent catastrophic system failure by allowing the machine to operate, though in a degraded mode, despite failure or malfunction of several integral units or subsystems.

HEDGE. A term that modifies fuzzy values.

HETEROASSOCIATIVE. A Neural Network where output patterns are distinct from input patterns.

HEURISTIC. Any rule of thumb, strategy, or technique used to limit the time required to search for solutions in large problem spaces.

HEURISTIC REASONING. Using an expert's rules of thumb, in the absence of precise control mechanisms, to reduce the space that must be searched for a solution.

HIDDEN LAYER. The middle layer in a Neural Network which takes outputs from the input layer, processes them, and passes them to the output layer.

HOPFIELD MODEL. One of the most commonly used types of Neural Networks; it is an associative memory network that acts as a filter.

INDUCTIVE. Learning by information repetition.

INFERENCE CHAIN. The path of reasoning that the inference engine follows to find a solution to the problem.

INFERENCE ENGINE. The part of an Artificial Intelligence system that uses knowledge in the knowledge base and acquired knowledge about the problem to form an expert solution.

INFERENCE MECHANISM. Controls the use of the knowledge base and databases when solving a problem.

INPUT LAYER. The first layer of neurons in a Neural Network. The input layer accepts the input values and passes them to the hidden layer. The input layer does not process data.

KNOWLEDGE ACQUISITION. The process of gathering the required information from various sources by a knowledge engineer.

KNOWLEDGE BASE. A form of coded knowledge about a specific domain.

KNOWLEDGE-BASED SYSTEM. A computer system whose usefulness derives primarily from a database containing human knowledge in a computerized format.

KNOWLEDGE ENGINEER. The person responsible for extracting, organizing, and encoding the knowledge related to the problem, and importing it into a knowledge base or database.

LABEL. A data item that serves to identify a data record, or a symbolic name used in a program to mark the location of a particular instruction or routine.

LAYER. A row of neurons in a Neural Network; there are usually three layers in a network.

LOGICAL SUM. The final unified result of the rule processes of a fuzzy system.

MACHINE LEARNING. The process or technique by which a device modifies its own behavior as the result of its past experience and performance.

MCCULLOCH-PITTS. A simple model of a neuron developed in 1943, it is a binary threshold unit that computes a weighted sum of its inputs from other neurons, and outputs a 1 or an 0, according to whether the sum is above or below a certain threshold.

META-KNOWLEDGE. Knowledge that reveals details about the system knowledge.

MODEL. A mathematical or physical system, obeying certain specified conditions, whose behavior is used to understand a physical, biological, or social system to which it is analogous in some way.

NEURAL NETWORK. A system modelled after the brain, used to solve problems by a mapping of input data to output data.

NEURO-FUZZY. Integrating a Neural Network and fuzzy logic to solve problems.

NEURON. The computational element in a Neural Network. A neuron computes the weighted sum of the inputs from other neurons and produces an output.

NODE. 1. Representation of an object that is related to other objects via connection weights. 2. The part of a Neural Network that calculates and processes information.

NOISE. Meaningless or erroneous bits that must be ignored or removed from a signal, especially in communication channels.

NON REAL-TIME LEARNING. Learning done in a Neural Network before attempting to solve application problems.

NONLINEARITY. A system in which the outputs do not correspond to the inputs in a direct or inversely proportional relationship.

OBJECT-ORIENTED. A representation form for knowledge in which all properties of an object are associated by the outgoing and incoming arcs at its node; the representation is geared towards manipulating objects as independent pieces of knowledge.

OPERATIONAL AMPLIFIER. A circuit element in which the output is determined by the difference of the input voltages.

OPPORTUNISTIC REASONING. The process of solving problems in a way that is not uniform or predictable. Opportunistic reasoning attempts to solve the problem in the best possible way for each individual situation. Rules fire when they are appropriate, without a priori ordering.

OUTPUT LAYER. The layer of neurons in a Neural Network that takes the outputs from the hidden layer, processes them, and then produces the output of the network.

PARALLEL PROCESSING. Performing many different calculations simultaneously to reach one conclusion or solution.

PERCEPTRON. A pattern recognition Neural Network in which the neurons connect only to the neurons in the next layer.

PREDICATE CALCULUS. A formal language of classical logic that uses functions and predicates to describe relations between individual entities.

PROBABILITY. A theory dealing with the uncertainty that results from random behavior. Probability is defined over the numeric range of 0 to 1.

PROCESSING ELEMENT. The part of the Neural Network where the computations are performed.

PRODUCTION RULES. A common technique for representing procedural knowledge in an Artificial Intelligence system.

REAL-TIME LEARNING. Learning done in a Neural Network while solving an application problem.

RECURSION. A technique in which an apparently circular process is used to perform an iterative process.

RULE-BASED. An Artificial Intelligence program where knowledge is represented as rules.

RULES. A method of representing knowledge in the form of IF THEN statements.

SCHEDULER. A part of an inference engine that determines which knowledge source should be activated, and in what order.

SCHMITT TRIGGER. A digital circuit element that uses feedback to hold the device at one of the two output states (on or off), unless a sufficiently large input is applied to overcome the feedback.

SEARCH TREE. The branches of possible solutions to a problem that start with initial information and spread out following different paths.

SELF-SUPERVISED. A method of training a Neural Network without the use of an external monitor. A feedback device that detects errors and adjusts the connection weights accordingly is used.

SEMANTIC NETWORK. A method of representing knowledge using nodes and arcs.

SLOTS. Subdivisions making up the frame that may contain data, procedures, or pointers to other frames.

SPURIOUS STATE. A point in a Neural Network that acts as an attractor, but is not one of the stored patterns.

STABILITY. The property of a system which remains under control and responds in a reasonable manner to an applied input.

SUPERVISED LEARNING. A method of training a Neural Network in which an external supervisor monitors the activities, and learning occurs on the basis of direct comparison of the output of the network with known correct answers.

SYMBOLIC PROCESSING. A method of processing knowledge where the relationships among the knowledge are stored using symbolic representations; thus, the system can deal freely with objects and not be concerned with their composition.

SYNAPSES. Pathways in a biological neuron that transmit pulses from one neuron to another.

SYNCHRONOUS. In step, or in phase, as applied to two or more circuits, devices, or machines.

TRADITIONAL PROGRAMMING. The use of standard programming languages, as opposed to application development languages, financial planning languages, query languages, and report programs.

TRAINING. A change in connection weight values of a Neural Network that results in capturing information that can be recalled later.

TRANSFER FUNCTION. The mathematical relationship between the output of a control system and its input: for a linear system, it is the Laplace transform of the output divided by the Laplace transform of the input under conditions of zero initial-energy storage.

TRANSPARENCY. A characteristic of knowledge in an Expert System referring to its independence from other knowledge. There is no processing information in the transparent knowledge.

TURING TEST. A method of determining the level of success of an Artificial Intelligence system based on a comparison with human experts.

UNSUPERVISED LEARNING. A method of training a Neural Network, where no external monitor is involved; the Neural Network organizes itself by grouping and generating its own classification of inputs.

VERY LARGE SCALE INTEGRATION. Describing Integrated Circuits with more than 1000 elements.

WEIGHT MATRIX. The collection of connection weights for an entire Neural Network.

INDEX